Robotair Full-Stack Intern Challenge

We are looking for motivated **Full-Stack Developer Interns** to join our Robotair development team with the possibility of a full-time job conversion. The challenge below will help us assess your skills in front-end, back-end development, GitHub usage, and deployment strategies.

Choose Your Task

Pick one of the two tasks below and submit the deliverables for evaluation.

Option 1: Robot Fleet Monitoring Dashboard

Objective: Build a **Fleet Monitoring Dashboard** to visualize the status and telemetry data of multiple robots.

Requirements:

1. **Features**:
    - Display a list of robots with the following details:
        - Robot ID (uuid)
        - Online/Offline status (Boolean)
        - Battery percentage (Int)
        - CPU usage (Int)
        - RAM consumption (Int)
        - Last updated timestamp (time)
        - Location coordinated (float,float)
    - Enable real-time updates using WebSockets or periodic polling (e.g., every 5 seconds).
    - Display a **map view** using libraries like **Leaflet.js** or **Mapbox** to visualize the robots' current positions.
    - Use the `fake_robot_data.json` provided for this challenge
2. **Backend**:
    - Use **FastAPI** or **Flask** to simulate data for upto 10 robots.
    - Generate mock telemetry data (battery, CPU, RAM, and position coordinates).
    - Expose a WebSocket/REST API for real-time updates.
3. **Frontend**:
    - Use **React.js** to display robot details in a clean, responsive dashboard.
    - Integrate a real-time **map view** showing the positions of the robots.
    - Highlight robots that are **offline** or have low battery levels (<20%).
4. **Bonus**:
    - Host the frontend on **Netlify** and the backend on **Heroku/Render**.
    - Containerize the app using **Docker**.
    - Add filters to display only robots based on their status (e.g., active, offline, low battery).

Option 2: ROS Log Viewer and Analyzer

Objective: Develop a **web-based ROS Log Viewer** to display and analyze logs generated by robots running ROS (Robot Operating System).

Requirements:

1. **Features**:
    - Upload a ROS log file (`.log` or `.txt` format).

    - Parse the log file to display key information, such as:
        - Timestamp
        - Log severity level (INFO, WARN, ERROR, DEBUG)
        - Node name (str)
        - Message content (str)
    - Allow filtering of logs by severity level (e.g., show only WARN or ERROR messages).
    - Include a **search bar** to find specific keywords in the logs.
    - Use the `fake_ros_logs.log` provided for this challenge
2. **Backend**:
    - Use **FastAPI** or **Flask** to parse the uploaded log files.
    - Implement a REST API to send parsed log data to the frontend.
3. **Frontend**:
    - Use **React.js** to display parsed logs in a table format.
    - Add filters and a search bar for better usability.
    - Highlight logs with **ERROR** or **WARN** severity levels for easy identification.
4. **Bonus**:
    - Host the frontend on **Netlify** and the backend on **Heroku/Render**.
    - Containerize the app using **Docker**.
    - Allow users to **download filtered logs** as a new file.

Deliverables

1. **GitHub Repository**:
    - A well-organized repository with clean commit history.
    - Include a detailed **README** with setup instructions, tools used, and features.
2. **Netlify Link** *(if applicable)*:
    - Host the frontend on Netlify and share the live link.
3. **Docker** *(if applicable)*:
    - Submit a `docker-compose.yml` or Dockerfile that containerizes the app.

Bonus Criteria

- **Deployment**: Hosting the app on Netlify (frontend) and Heroku/Render (backend).
- **Containerization**: Demonstrate the ability to containerize the entire application using Docker.

If you complete either bonus task (Netlify hosting or Docker containerization), you will advance **directly to the final interview**!

Evaluation Criteria

1. **UI/UX Design**: User-friendly and visually appealing interface.
2. **Code Quality**: Clean, modular, and well-documented code.
3. **Git Usage**: Effective version control with clear commits and branches.
4. **Backend**: Functional and efficient Python backend.
5. **Deployment/Containerization** *(Bonus)*: Ability to deploy or containerize the app.

How to Submit

- Share the GitHub repository link.
- Include the Netlify live link *(if applicable)*.
- Include any Docker files in the repository *(if applicable)*.

This challenge is designed to test your **technical expertise**, **problem-solving abilities**, and **creativity** while ensuring the solution aligns with real-world robotics applications. We look forward to seeing your solutions! Good luck!