Carbonleap : Full Stack Intern

Take-Home Assignment: Field Insights Dashboard

Assignment Overview

You will build a **Field Insights Dashboard**—an interactive web app designed to simulate, ingest, and analyze farm sensor data (soil moisture, temperature, etc.). The system will allow users to upload or submit sensor readings, process data both synchronously and in the background, and display analytics/results in a minimal UI.

Your solution **must cover**:

- RESTful API design (Python/FastAPI)
- Background processing: **Celery** (for stats, aggregation, or batch tasks)
- Async/threaded processing (for parallel uploads/calculations)
- Minimal frontend UI (React, or template-based) to show end-to-end flow, with a responsive design that works on both desktop and mobile viewports
- Clean, modular, and well-architected code (bonus: Docker)
- Cloud-hosted submission

Assignment Instructions

1. Backend

- **Tech:** Python (FastAPI), PostgreSQL
- **Features:**
    - API endpoint(s) to post sensor data (JSON for different "fields" and "sensor types")
    - API endpoint to fetch analytics/statistics (e.g., average soil moisture, trends by hour/field)

  - Celery worker(s) to run background data aggregation or analytics periodically/on-demand
  - Async/threaded processing for handling bursts of uploads or to offload CPU-bound computations
  - Persist data in PostgreSQL

2. Frontend

- **Tech:** React (preferred), or template UI
- **Features:**
  - Form/component to submit or upload sensor data/batches (simulate upload bursts)
  - Dashboard to view latest analytics/stats (fetch from backend API)
  - Job/status view or notification to show background processing state
  - **Must be responsive** — the UI should look and work well on desktop and mobile devices

3. Architecture & Code

- Prioritize clean, modular, *well-commented* code.
- Organize backend/frontend codebases logically
- Include a README file with your deployment process.

Guidelines for Sensor Data Generation

To realistically stress-test your backend and demo meaningful analytics, **generate a larger, more comprehensive sensor dataset** beyond just 5–10 rows.

Options to Generate Data

- **Manual (small scale):** Handcraft CSV/JSON files in the described schema.
- **LLM-powered (recommended, bulk):** Ask LLMs like ChatGPT/Gemini to generate hundreds/thousands of sensor records using the schema below.

Example LLM Prompt

> "Generate 500+ farm sensor readings as a table or CSV. Each reading must have:
>
> - timestamp (ISO 8601, UTC),
> - field_id (e.g., 'field_1', 'field_2', ...),
> - sensor_type (one of: soil_moisture, temperature, humidity, ph, sunlight, rainfall, wind_speed, soil_nitrogen),
> - reading_value (in a realistic range for the type),
> - unit (see below).
>   Output as CSV or JSONL."

Recommended Data Schema

Column	Type	Example	Description
timestamp	string	2025-07-30T08:15:20Z	ISO8601, UTC
field_id	string	field_2	Identifies the farm field
sensor_type	string	soil_moisture	See list below
reading_value	float	28.7	See prescribed value ranges
unit	string	%	See list below

Supported Sensor Types

sensor_type	reading_value range	unit	Description
soil_moisture	0–100	%	Soil moisture content
temperature	-10–45	°C	Air temperature
humidity	0–100	%	Air relative humidity
ph	4.5–9	pH units	Soil acidity/alkalinity
sunlight	0–1,500	W/m²	Solar radiation
rainfall	0–50	mm	Rainfall amount
wind_speed	0–30	m/s	Wind speed

sensor_type	reading_value range	unit	Description
soil_nitrogen	0–100	mg/kg	Soil nitrogen content

Sample Sensor Records (JSON)

```
[
  {
    "timestamp": "2025-07-30T08:15:20Z",
    "field_id": "field_2",
    "sensor_type": "humidity",
    "reading_value": 55.2,
    "unit": "%"
  },
  {
    "timestamp": "2025-07-30T08:16:50Z",
    "field_id": "field_2",
    "sensor_type": "ph",
    "reading_value": 6.8,
    "unit": "pH units"
  }
]
```

Tips:

- Mix sensor types in your dataset—each field can have multiple sensor readings per timestamp.
- Generate at least 250–1,000 records for realistic evaluation of background and parallel processing.
- Include your generated data file and describe *how* you generated it (prompt, script, etc.) in your submission.

Tech Requirements

- **Backend:** Python 3.8+, FastAPI, PostgreSQL, Celery, (bonus: Redis, Docker)

- **Frontend:** React, Axios/Fetch (template UI allowed), must be responsive for mobile
- **Task Processing:** Celery (background), Python threading/asyncio (parallel uploads)
- **Version Control:** GitHub public repo

Evaluation Criteria

- Well-structured, RESTful API implementation
- Correct Celery + async/threaded job usage
- Minimal but functional frontend demonstrating API and jobs
- Modular, maintainable code
- Live demo hosted and working
- Creativity, completeness, and quality of analytics

Submission Guidelines

1. **Push all code** (backend & frontend) to a public GitHub repo.
2. **Host a live demo** (using any free hosting).
3. **Upload URL and repo link** via the provided Google Form.
4. **Include generated data file and how you created it.**
5. Recommended work time: 6–8 focused hours. Submit within 48 hours of assignment.
6. Note any blockers or service issues in your submission notes.

We're excited to see your approach to this real-world challenge—good luck!