Backend Intern Take-Home Assignment: HumanChain AI Safety Incident Log API

Goal: To assess your fundamental backend development skills, including API design, request handling, and data persistence, by building a simple service relevant to HumanChain's mission in AI safety.

Context: HumanChain is a deep-tech software AI startup at the forefront of AI safety, aiming to build a safer, more trustworthy, and human-centric digital world. This assignment involves creating a basic backend API service to log and manage hypothetical AI safety incidents.

Instructions:

1. **Choose ONE Language/Framework:** Select a backend language and framework you are comfortable with.
    - **Languages:** Python, Ruby, Rust, Typescript
    - **Frameworks:** Django, Flask, Ruby on Rails, Node.js (with Express, etc.), Next.js
    - Choose a combination you know well (e.g., Python with Django/Flask, Node.js with Express, Ruby on Rails, Rust with Actix/Axum/Rocket, etc.). Others are acceptable if you specify them clearly.
2. **Complete the Task:** Implement the specific API task outlined below.
3. **Focus:** Prioritize clean, readable, well-structured code and functional API endpoints. Implement the requested features, including routing, request/response handling (JSON), data manipulation, and persistence using a database. Adherence to basic REST principles is expected.
4. **Data:** Use a database to store and manage the AI Safety Incidents. You may choose any database you are comfortable with (e.g., PostgreSQL, MySQL, SQLite, Supabase, MongoDB, etc.). You are also free to use an Object-Relational Mapper (ORM) or Object-Document Mapper (ODM) suitable for your chosen language/framework and database.
5. **Submission:**
    - Package your code in a zip file or provide a link to a public Git repository (e.g., GitHub, GitLab).
    - Include a README.md file with:
        - Clear instructions on how to build/install dependencies and run your project locally.
        - The language/framework choice.
        - Clear instructions on setting up the required database schema and configuring the database connection (e.g., environment variables,

configuration files, required setup scripts).
    - Examples of how to call each API endpoint (e.g., using curl or mentioning the path and HTTP method).
  - (Optional) Briefly mention any design decisions or challenges in the README.

API Task: AI Safety Incident Log Service

- **Task:** Create a simple RESTful API service to log and manage hypothetical AI safety incidents stored in a database.
- **Data Structure:** Each incident should have at least the following fields stored in the database:
  - id: A unique identifier (database-generated primary key is recommended).
  - title: A short summary of the incident (string).
  - description: A more detailed description of the incident (string/text).
  - severity: The assessed severity level (e.g., "Low", "Medium", "High") (string).
  - reported_at: Timestamp when the incident was logged (datetime/timestamp).
- **Initial Data:** Your setup instructions in the README should include how to optionally pre-populate the database with 2-3 sample incidents (e.g., via a migration script or manual steps).
- **Required API Endpoints:** Implement the following endpoints, ensuring they handle JSON requests and responses where applicable and interact with the database:
  1. **GET /incidents**
     - **Action:** Retrieve all incidents currently stored in the database.
     - **Response:** 200 OK with a JSON array of incident objects.
       [
        { "id": 1, "title": "...", "description": "...", "severity": "...", "reported_at": "..." },
        { "id": 2, "title": "...", "description": "...", "severity": "...", "reported_at": "..." }
       ]

  2. **POST /incidents**
     - **Action:** Log a new incident to the database. The database should assign a unique ID, and the reported_at timestamp should be set automatically upon creation.
     - **Request Body:** JSON object containing title, description, and severity.
       { "title": "New Incident Title", "description": "Detailed description here.", "severity": "Medium" }

     - **Response:** 201 Created with the JSON object of the newly created incident (including its assigned id and reported_at). Include basic

validation (e.g., ensure required fields are provided and severity is one of the allowed values). If validation fails, return an appropriate 400 Bad Request response.
{ "id": 3, "title": "New Incident Title", "description": "Detailed description here.", "severity": "Medium", "reported_at": "2025-04-02T18:00:00Z" }

3. **GET /incidents/{id}**
    - **Action:** Retrieve a specific incident by its id.
    - **Path Parameter:** {id} representing the unique ID of the incident.
    - **Response:** 200 OK with a JSON object representing the requested incident. Returns 404 Not Found if no incident with the given id exists.
      { "id": 1, "title": "...", "description": "...", "severity": "...", "reported_at": "..." }

4. **DELETE /incidents/{id}**
    - **Action:** Delete the incident with the specified id from the database.
    - **Path Parameter:** {id} representing the unique ID of the incident to delete.
    - **Response:**
        - 204 No Content or 200 OK (with a confirmation message) if deletion is successful.
        - 404 Not Found if no incident with the given id exists in the database.

- **Error Handling:** Implement basic error handling for potential issues.

Good luck! We look forward to seeing your work.