1. Project Overview

- **Project Name**: HumanChain AI Safety Incident Log API
- **Description**: 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 backend API service with a minimal frontend for logging and managing AI safety incident."

2. Technologies & Dependencies

- **Frontend (HTML/CSS/JS)**

  Purpose: Minimal UI to add, filter, and manage incidents.

  Key Features:

  - Form to submit new incidents (index.html).
  - Dynamic rendering of incidents with filter buttons (main.js).

- **Backend (Node.js/Express.js)**

  Core Dependencies:

  - express: Framework for API routing and middleware.
  - mongoose: MongoDB ODM for schema modeling and queries.
  - cors: Enables cross-origin requests for frontend-backend communication.
  - dotenv: Loads environment variables (e.g., MONGODB_URI).
  - morgan: Logs HTTP requests for debugging.

- **Database (MongoDB)**

  Purpose: Stores incident data with auto-generated _id and timestamps.

  Schema: Tracks title, description, category (severity), and completed status.

Why This Stack?

- **Express.js**: Lightweight, RESTful API setup.
- **MongoDB**: NoSQL flexibility for incident logging.

- **Minimal Frontend**: Demonstrates API integration without complex frameworks.

3.Installation: (Go through ReadMe file for detailed explanation)

Github: https://github.com/manojkadali/HumanChain---AI-Safety-Incident-Log-API

- Prerequisites

1. install Node.js v14 or higher(install from nodejs.org).
2. npm usally comes with nodejs,if not install it
3. install mongoDB Community edition with compass(GUI), or mongoDB Atlas connection string.

- Clone the repository or download the zip file.
- open terminal for that project folder ->npm install
- Configure environment variables by creating .env file
  ```
   PORT=3000
  MONGODB_URI=mongodb://localhost:27017/incident-log
  ```
- Establish connection with mongodb compass by giving the same URI in MongoDB
- Start the server : ->npm run rev
- Open the browser and go to localhost:3000

4.Technical Architecture

Folder Structure:

```
project-root/
 ├── config/      # Database configuration
 ├── controllers/  # request handlers logic
 ├── models/      # MongoDB schemas
 ├── public/      # Frontend (HTML/CSS/JS)
 ├── routes/      # API endpoint definitions
 └── server.js    # Entry point
```

MVC Breakdown

- **Model: models/item.js (Defines the incident schema)**
- **View: public/index.html (Minimal frontend)**
- **Controller: controllers/itemController.js (Handles CRUD operations)**
- **Routes: routes/api.js (Maps endpoints to controllers)**

5. Data Model

Incident Schema (models/db.js) | Field | Type | Required | Description |

Field	**Type**	**Required**	**Description**
id	**ObjectId**	**Yes**	**Unique identifier (auto-generated)**
title	**String**	**Yes**	**Short summary of the incident**
description	**String**	**Yes**	**Detailed description of the incident**
severity	**String**	**Yes**	**"Low", "Medium", or "High"**
created_at	**Date**	**Yes**	**Timestamp of incident creation**

6. Workflow & Data Flow

1. Client Request: HTTP client sends request (e.g., POST /incidents with JSON payload).
2. Routing: routes/api.js matches the path and HTTP method, forwarding request to corresponding controller in incidentController.js.
3. Controller Logic:
   - Validation: Check required fields (title, description, severity) and that severity ∈ {"Low","Medium","High"}.
   - Model Interaction: Use Mongoose model (Incident) to query or mutate the MongoDB database.

4. Database: MongoDB stores incidents in the incidents collection; Mongoose handles schema enforcement and timestamps.
5. Response: Controller sends JSON response with appropriate HTTP status codes.
6. Error Handling: Errors bubble to errorHandler.js middleware, formatting a JSON error response.

7. API Endpoints

Method	Path	Description	Controller Function
GET	/items	Retrieve all incidents	getItems
POST	/items	Create a new incident	createItem
GET	/items/:id	Retrieve incident by ID	getItem
DELETE	/items/:id	Delete an incident by ID	deleteItem

8. Controller File Description (controllers/ItemController.js)

Function Name	Purpose & Logic Summary
getItems	Uses Item.find() to fetch all incidents, sorts by reported_at descending, and returns them with HTTP 200. Errors forwarded to error middleware.
createItem	Extracts title, description, severity from req.body, validates inputs (presence and allowed severity), constructs a new Incident, saves it to MongoDB (auto-assigns id and reported_at), and returns it with HTTP 201. Validation failures return HTTP 400.

getItem	**Reads id from req.params, uses Item.findById(id) to retrieve the document. If found, returns it with HTTP 200; otherwise returns HTTP 404. Errors forwarded to middleware.**
deleteItem	**Reads id from req.params, uses Item.findByIdAndDelete(id) to remove the document. If deletion succeeds, returns HTTP 204; if no document found, returns HTTP 404. Errors forwarded onward.**

9. Error Handling

- **Validation Errors**: Return 400 Bad Request with descriptive message.
- **Not Found:** Return 404 Not Found when querying or deleting non-existent IDs.
- **Server Errors:** Uncaught exceptions forwarded to errorHandler.js middleware, returning 500 Internal Server Error.

10. Environment & Configuration

- **Environment Variables (in .env):**
  - MONGODB_URI: Connection string for MongoDB Atlas or local instance.
  - PORT: Port for Express server (default: 3000).

VIDEO Explanations:

Local setup Video for HumanChain Project:
https://drive.google.com/file/d/1TG9c5JdXTtdS45OdyUzDqaWtlH_25ne6/view?usp=sharing

Project Overview video:

drive.google.com/file/d/1EbEbMy3MDzWL9CZGFA90halfPEIQk33k/view?usp=sharing

PostMan API Testing:
https://drive.google.com/file/d/1EbEbMy3MDzWL9CZGFA90halfPEIQk33k/view?usp=sharin
g