Real-Time Incident Monitoring Exercise

Deliverable: A public GitHub repository

Overview

Create a small **full-stack application** consisting of a backend service and a web-based UI.

The system simulates a **real-time incident monitoring platform** (e.g. operational alerts, system events, or tickets). The goal is to evaluate:

- Basic UI skills (HTML, JavaScript)
- API consumption from the browser
- Real-time data transfer (WebSockets)
- Backend design with modern Node.js
- Docker and multi-container orchestration

The application does not need to be production-ready, but it should be **clean, readable, and easy to run**.

Functional Requirements

Domain Model: Incident

An *Incident* represents a system event that operators need to be aware of.

Incident fields:

- `id` (string or UUID)
- `title` (string)
- `severity` (string, allowed values: `low`, `medium`, `high`)
- `status` (string, allowed values: `open`, `acknowledged`, `resolved`)
- `createdAt` (timestamp)

Backend Requirements

REST API

Implement a REST API with the following endpoints:

1. Create an Incident

`POST /incidents`

Creates a new incident.

The backend should:

- Validate input
- Set default `status` to `open`
- Set `createdAt` automatically

---

2. List Incidents

`GET /incidents`

Returns all incidents, ordered by creation time (newest first).

---

3. Update Incident Status

`PATCH /incidents/:id`

Allows updating the incident `status`.

Real-Time Updates

The backend must expose a **WebSocket** endpoint.

Whenever an incident is:

- created
- or updated

The server should broadcast an event to all connected clients with the updated incident data.

You may use:

- native WebSocket support
- or a lightweight library (e.g. `ws`)

Frontend Requirements

Create a simple UI using **plain HTML and JavaScript** (no frameworks required).

UI Features

The UI should:

1. Display a list/table of incidents
2. Show title, severity, status, and creation time
3. Allow creating a new incident via a form
4. Allow updating the status of an incident (e.g. buttons or dropdown)
5. Update in **real time** when incidents are created or changed (via WebSockets)

The UI does not need to be visually polished, but it should be:

- usable
- reasonably structured
- easy to understand

Technical Requirements

Your solution must use:

Backend

- **Node.js 24+**
- Any framework (or none)
- ES module syntax (`import` / `export`)
- async / await

Frontend

- HTML
- Vanilla JavaScript
- Fetch API for REST calls
- WebSocket API for real-time updates

Infrastructure

- Docker
- Separate containers for:
  - backend
  - frontend
  - database (mongodb)

- `docker-compose` to run the full system

README Requirements

Include a `README.md` with:

- Project overview
- How to build the Docker images
- How to start the system using `docker-compose`
- Which ports are exposed
- Any assumptions or design decisions

Optional (Nice to Have)

These are not required, but appreciated if included:

- Input validation with clear error messages
- Simple logging

General Notes

- No authentication is required
- Keep the solution simple and readable
- You may use any libraries you find appropriate, but the less the better.
- Especially if you can avoid the use of any frameworks, this would be considered a plus
- Do not over-engineer the solution
- No really need to focus on the CSS and design, even the default styling is fine
- Please use a dockerized MongoDB for persistence

What We Care About

We are interested in:

- Clear and maintainable code
- Correct use of APIs and WebSockets
- Understanding of client/server interaction
- Practical Docker usage

- Overall developer experience
- A solution that is not relied on frameworks or heavy libraries

---

Submission

Please share:

- A link to your public GitHub repository