System Assumptions and Data Schema for Downtime Incident Triage Assistant Agent

Assumptions

1. Input & Data Format

- All incidents are provided in a structured JSON format (not natural language).
- Timestamps follow ISO 8601 (`YYYY-MM-DDTHH:MM:SSZ`).
- If `measured_value` or `threshold` are not applicable, they are set to `null`.
- API endpoint expects a flat JSON payload (fields at the root level, not nested under `"new_incident"`).

2. System Context

2.1 Sensor Infrastructure

- Each machine on the production line is equipped with IoT sensors capable of detecting abnormal operating conditions (e.g., temperature, pressure, conveyor jams, leaks).
- These sensors transmit real-time data to a centralized monitoring system (such as SCADA**,** PLC, or plant management software).

2.2 Centralized Incident Generation

- The central monitoring system continuously analyzes incoming raw sensor data.
- When a threshold breach or abnormal condition is detected, it generates a structured incident JSON containing:
  - `incident_id`
  - `machine`
  - `timestamp`
  - `type`
  - `measured_value`
  - `threshold`
  - `status`
- Only incident JSONs are forwarded to the AI agent — the agent never interacts directly with raw sensor streams.

2.3 Communication and Reliability

- Communication between the central system and the AI agent is assumed to be reliable and near real-time**.**
- Each request sent to the agent contains a single incident JSON**.**

2.4 Historical Data Availability

- A vector database of past incidents exists, pre-populated with historical incident records.
- Past incidents follow the same structured schema (or are converted into it before storage).
- Retrieval is based on similarity (e.g., machine type, incident type, status, and other relevant fields).

2.5 Scope Boundaries

- The system's scope begins after an incident JSON is generated**.**
- Pre-incident processes such as raw sensor calibration, noise filtering, or alert threshold tuning in the monitoring system are out of scope**.**
- The AI agent's responsibility is only:
    - Compare the new incident against historical data,
    - Provide a risk assessment**,**
    - Identify probable root causes**,** and
    - Suggest recommended actions**.**

3. Retrieval & Historical Data

- Vector search considers all retrieved incidents**,** not just one.
- Analysis aggregates causes and actions from multiple incidents when possible.
- Historical data is assumed to be accurate, consistent, and relevant**.**

4. AI Analysis & Output

- Risk levels are standardized as High, Moderate, or Low**.**
- The AI always bases recommendations on retrieved past incidents only (no hallucinated causes/actions).
- Outputs follow a consistent structured format for downstream integration.

Schema

1. Input incident schema

1.1 JSON Schema (Structure)

Field	Type	Type	Description
incident_id	string	Yes	Unique identifier for the incident (e.g., `AUTO-INC-002`).
machine	string	Yes	Machine/line where the incident occurred (e.g., *Filler Line B*).
timestamp	string	Yes	Time of the incident in ISO 8601 format (UTC).
type	string	Yes	Type of incident (e.g., `ConveyorJam`, `Overheat`, `Leakage`).
measured_value	number | null	Optional	Sensor reading at incident time (e.g., `85.0`). Can be null.
threshold	number | null	Optional	Threshold limit that triggered the incident. Can be null.
status	string	Yes	System response (`emergency_stop`, `warning`, `shutdown`).

1.2 Example Input

{

"incident_id": "AUTO-INC-002",

"machine": "Filler Line B",

"timestamp": "2025-08-18T09:45:00Z",

"type": "ConveyorJam",

"measured_value": null,

"threshold": null,

"status": "emergency_stop"

}

2. Output Schema

2.1 JSON Schema (Structure)

Field	Type	Description
incident	string	Flattened representation of the new incident.
risk_level	string	Risk classification: `High`, `Medium`, or `Low`.
root_causes	[string]	List of possible root causes derived from past incidents.
recommended_actions	[string]	List of recommended actions corresponding to the identified root causes.
relevant_past_incidents_count	integer	Number of past incidents retrieved and considered in analysis.

2.2 Example Output


```
{
  "incident": "incident_id: AUTO-INC-002 | machine: Filler Line B | timestamp:
2025-08-18T09:45:00Z | type: ConveyorJam | measured_value: None | threshold: None |
status: emergency_stop",
  "risk_level": "High",
  "root_causes": [
    "Misaligned guide rail",
    "Loose hose connection"
  ],
  "recommended_actions": [
    "Adjust the misaligned guide rail",
    "Tighten the loose hose connection",
    "Clean up any spillage and ensure safety measures are followed"
  ],
  "relevant_past_incidents_count": 2
}
```

3. Schema Validation with Pydantic

Used Pydantic models (via FastAPI) to enforce schema validation on input incidents.

```python
from pydantic import BaseModel
from typing import Optional

class Incident(BaseModel):
    incident_id: str
    machine: str
    timestamp: str
    type: str
    measured_value: Optional[float] = None
    threshold: Optional[float] = None
    status: str
```