Aqua-Lert Backend Documentation

1. Project Overview

This project is a **Smart Water Leakage Detection API** built using **Python, FastAPI, SQLAlchemy**, and a **MySQL** database. The system monitors water flow through a pipeline network with sensors and generates alerts when leaks, anomalies, or low battery levels are detected. Unlike the earlier linear comparison approach, the backend now supports **pipeline topology** with parent-child relationships for accurate leak localization in **branching structures**.

The project is structured into the following Python files:

- **main.py**: Main FastAPI application with all routes.
- **database.py**: Database connection setup using SQLAlchemy.
- **models.py**: ORM models representing database tables.
- **utils.py**: Utility functions for processing sensor data and generating alerts.
- **schemas.py**: Pydantic models for input/output validation.

---

2. Setup Instructions

1. Install dependencies:

```
pip install fastapi sqlalchemy pymysql pydantic uvicorn
```

1. Configure database credentials in `database.py`.

2. Run the FastAPI server:

```
uvicorn main:app --reload
```

1. The server will be available at `http://127.0.0.1:8000`.

---

3. Database (database.py)

Purpose: Establishes a connection to MySQL via SQLAlchemy.

- **DATABASE_URL**: Connection string in format `mysql+pymysql://user:password@host:port/db`.
- **engine**: SQLAlchemy engine object.
- **SessionLocal**: Factory for database sessions.

- **Base**: Declarative base class for models.
- **get_db()**: Provides a DB session for API routes.

Usage: All routes depend on `get_db()` for DB access.

---

4. Models (models.py)

Purpose: ORM classes that map to database tables.

Enum Classes

- **SensorStatus**: `active`, `inactive`, `maintenance`
- **AlertType**: `leak`, `anomaly`, `low_battery`
- **Severity**: `low`, `medium`, `high`
- **AlertStatus**: `active`, `resolved`

Tables

4.1 Sensor

- `sensor_id`: Primary key (string)
- `location`: Location of sensor
- `pipe_diameter_mm`: Pipe diameter in mm
- `install_date`: Installation date
- `status`: Enum field
- ``: Defines topology by linking to another sensor

4.2 SensorData

- `id`: Primary key (auto-increment)
- `sensor_id`: FK → Sensor
- `timestamp`: Time of reading
- `flow_rate`: Flow measurement (Decimal)
- `battery_level`: Sensor battery level (int)

4.3 ProcessedData

- `id`: Primary key
- `sensor_id`: FK → Sensor
- `timestamp`: Time of processed data
- `smoothed_flow`: Averaged flow value
- `flow_diff`: Difference compared to previous values

4.4 Alert

- `alert_id`: Primary key

- `sensor_from`: Origin sensor
- `sensor_to`: Destination/child sensor (optional)
- `timestamp`: Time of alert
- `alert_type`: Enum (leak, anomaly, etc.)
- `severity`: Enum
- `probability`: Decimal probability value
- `status`: Enum (active, resolved)

---

5. Utilities (utils.py)

Purpose: Core logic for leakage detection and alert generation.

5.1 Sigmoid Function

```
def sigmoid(x, x0=0, k=1):
    return 1 / (1 + exp(-k*(x-x0)))
```

Smooth mapping from real values to range `[0, 1]`. Used to model probability curves.

5.2 Leak Probability Calculation

```
def compute_leak_probability_sigmoid(flow1, flow2, battery1, battery2):
    # returns probability [0,100]
```

- Inputs: parent and child sensor flows, battery levels
- Outputs: Leak probability %
- Combines `flow_avg`, `flow_diff`, and `battery` using sigmoid functions

5.3 Process Sensor Data Topology

```
def process_sensor_data_topology(db, sensors, new_readings):
    # process readings based on parent-child relationships
```

- Iterates through sensors
- Builds topology from `parent_sensor_id`
- Compares parent flow with **sum of child flows**
- Generates Alerts for:
- **Leak** (deviation detected)
- **Anomaly** (sudden drops)
- **Low Battery** (below threshold)
- Stores processed results in DB

Inputs:

- `sensors`: list of Sensor ORM objects
- `new_readings`: dict `{sensor_id: SensorData}`

Outputs:

- List of Alert objects

---

6. Main API (main.py)

Purpose: Exposes all routes for managing sensors, data, and alerts.

6.1 Dependency

```
def get_db(): ...
```

Provides DB session per request.

6.2 Sensor Routes

- **POST /sensors** → Register sensor (includes `parent_sensor_id`)
- **GET /sensors** → List sensors
- **PUT /sensors/{id}** → Update sensor details
- **DELETE /sensors/{id}** → Delete sensor & related data

6.3 Sensor Data Routes

- **POST /sensors/{id}/data** → Insert single sensor reading
- **POST /sensors/data** → Insert batch readings
- **GET /sensors/{id}/data** → Get recent readings
- **PUT /sensors/{id}/data/{data_id}** → Update reading
- **DELETE /sensors/{id}/data/{data_id}** → Delete reading
- **GET /sensors/{id}/data/filter** → Filter readings by date range

6.4 Alert Routes

- **GET /alerts** → Fetch active alerts
- **POST /alerts/resolve/{id}** → Resolve a single alert
- **PUT /alerts/{id}** → Update severity/status
- **DELETE /alerts/{id}** → Delete alert
- **GET /alerts/filter** → Filter by sensor/type/severity/status
- **POST /alerts/resolve/bulk** → Resolve multiple alerts

6.5 Alert Logic

- **Leak detection**: Based on parent vs child flow comparison
- **Low battery**: < 20%
- **Anomaly**: sudden negative drops in flow

---

7. Input/Output Examples

Register Sensor

Request:

```
POST /sensors?
sensor_id=S2&location=Pipe2&pipe_diameter_mm=50&parent_sensor_id=S1
```

Response:

```
{ "message": "Sensor registered successfully", "sensor": "S2" }
```

Insert Batch Data

Request:

```
{
  "readings": [
    {"sensor_id": "S1", "flow_rate": 200.0, "battery_level": 90, "timestamp":
"2025-09-07T10:00:00"},
    {"sensor_id": "S2", "flow_rate": 120.0, "battery_level": 88, "timestamp":
"2025-09-07T10:00:00"}
  ]
}
```

Response:

```
{
  "alerts_generated": [
    {"alert_id": 5, "sensor_from": "S1", "sensor_to": "S2", "alert_type":
"leak", "severity": "high", "probability": 91.2, "timestamp":
"2025-09-07T10:00:05", "status": "active"}
  ]
}
```

Fetch Alerts

```
GET /alerts
```

Response:

```
[
  { "alert_id": 5, "sensor_from": "S1", "sensor_to": "S2", "alert_type":
"leak", "severity": "high", "probability": 91.2, "status": "active" }
]
```

---

8. How It Works

1. **Sensors** are registered with topology (`parent_sensor_id`).
2. **Sensor data** is sent via API (single or batch).
3. **Processing**:
4. Stores raw readings in DB
5. Computes smoothed values
6. Runs **topology-based leak detection**
7. **Alerts** are generated (leak, anomaly, low battery).
8. **Alerts** are stored and available via API for frontend or notifications.

End of Updated Backend Documentation