Project Overview

- Backend consists of three micro services:

Signing Micro-service {done}:

- Handles user authentication and authorisation.
- Technologies: FastApi, PostgreSQL, Redis.

Device Management Micro-service:

- Manages device registration, configuration, and status.
- Technologies: FastApi, PostgreSQL, Redis.

Monitoring Micro-service:

- Collects and visualises data from IoT devices.
- Technologies: FastApi, MongoDB, Socket.IO.

Additionally, **Docker**, **Kubernetes**, **RabbitMQ**, and other tools will be used to orchestrate and manage the micro services.

- Front-end : Feel free to choose your preferred technologies stack

Final deploy on AWS, AZURE or GC will be appreciated

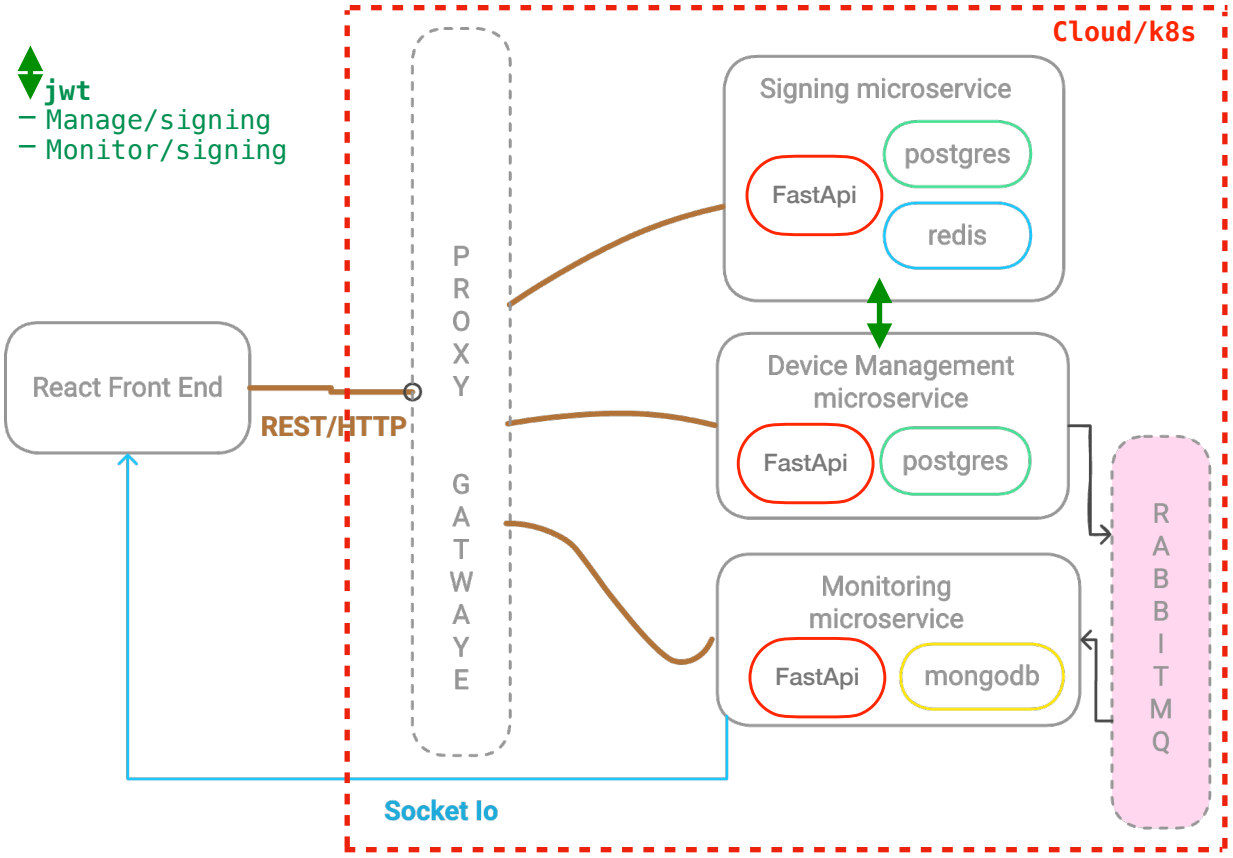

Fig1.1 Architecture of the Project

Microservices Communication

Signing & Device Management:

- Use **HTTP REST** for synchronous communication between these services.

Device Management & Monitoring:

- Use **RabbitMQ** for asynchronous event-driven communication to stream real-time IoT data to the monitoring microservice.

Monitoring:

- Use **Socket.IO** for real-time data updates to the clients.

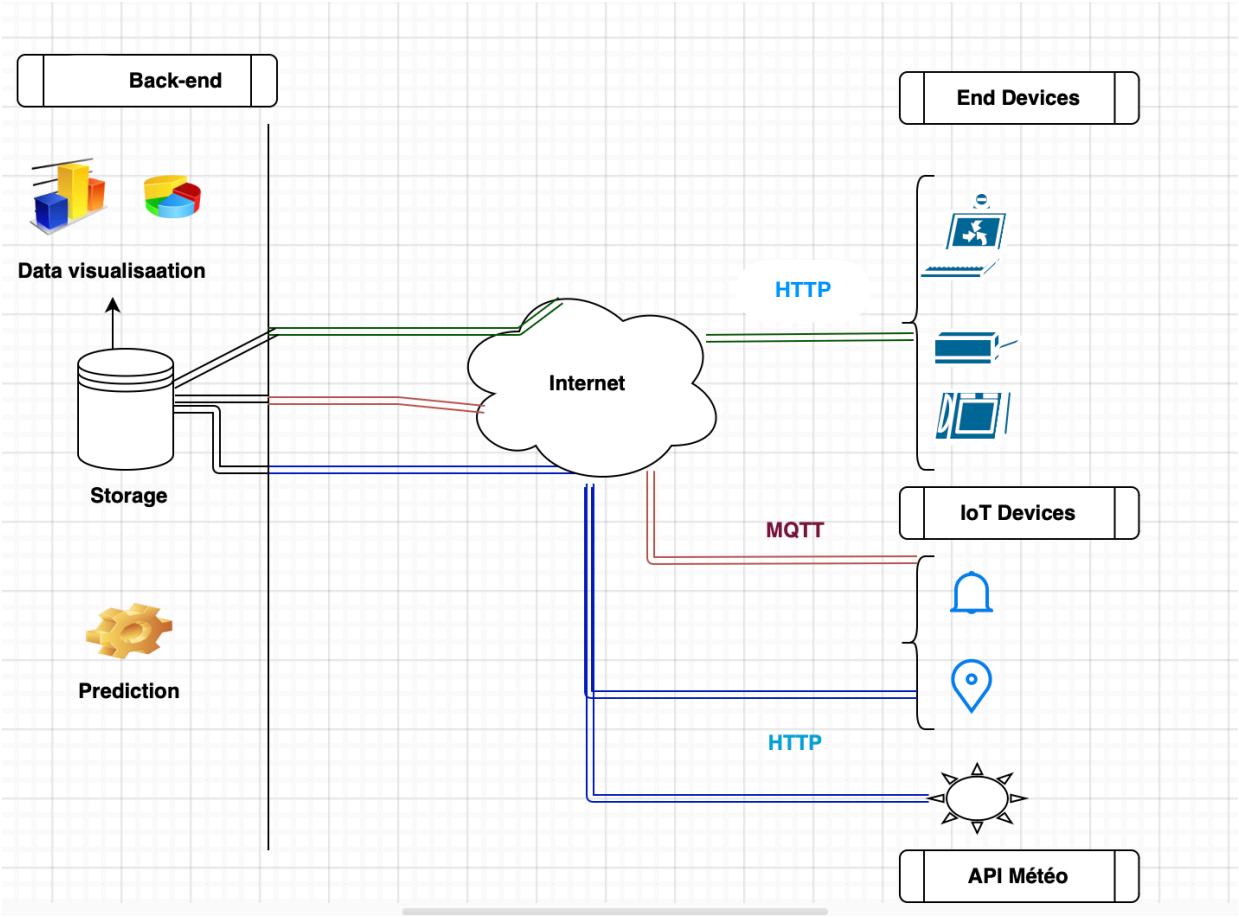

Fig 1.2 Gathering Information from Iot

Data Flow

- Devices send data to the **Device Management** micro service.
- The **Device Management** micro service pushes relevant data/events to the **Monitoring** micro service via RabbitMQ.
- The **Monitoring** micro service stores and serves data through FastApi and MongoDB and provides real-time updates using Socket.IO.

Technologies

- **Microservices Framework**:
  - **FastAPi**: Lightweight framework for each micro service.
- **Database**:
  - **PostgreSQL**: Relational database for Signing and Device Management services.
  - **Redis**: Caching for fast access and session management in Signing and Device Management.
  - **MongoDB**: Document-oriented database for storing unstructured IoT data in Monitoring.

- **Messaging System**:
  - **RabbitMQ**: Message broker for asynchronous communication between Device Management and Monitoring services.

- **Real-Time Communication**:
  - **Socket.IO**: Enables real-time data push from the Monitoring service to clients (front-end).
- **Orchestration**:
  - **Kubernetes (microk8s**: Manages containerized micro services for scalability and high availability.
- **Containerization**:
  - **Docker**: Package each micro service into containers for consistent environments.
- **API Gateway**:
  - **nginx** : Acts as a single entry point for external clients, routing requests to appropriate microservices.
- **Monitoring & Logging** : if used will be appreciated
  - **Prometheus**: Monitor application performance and resource usage.
  - **Grafana**: Visualize metrics.
- **IoT Data Simulation**:
  - Use **MQTT** and custom Python scripts to simulate IoT device data {done}.

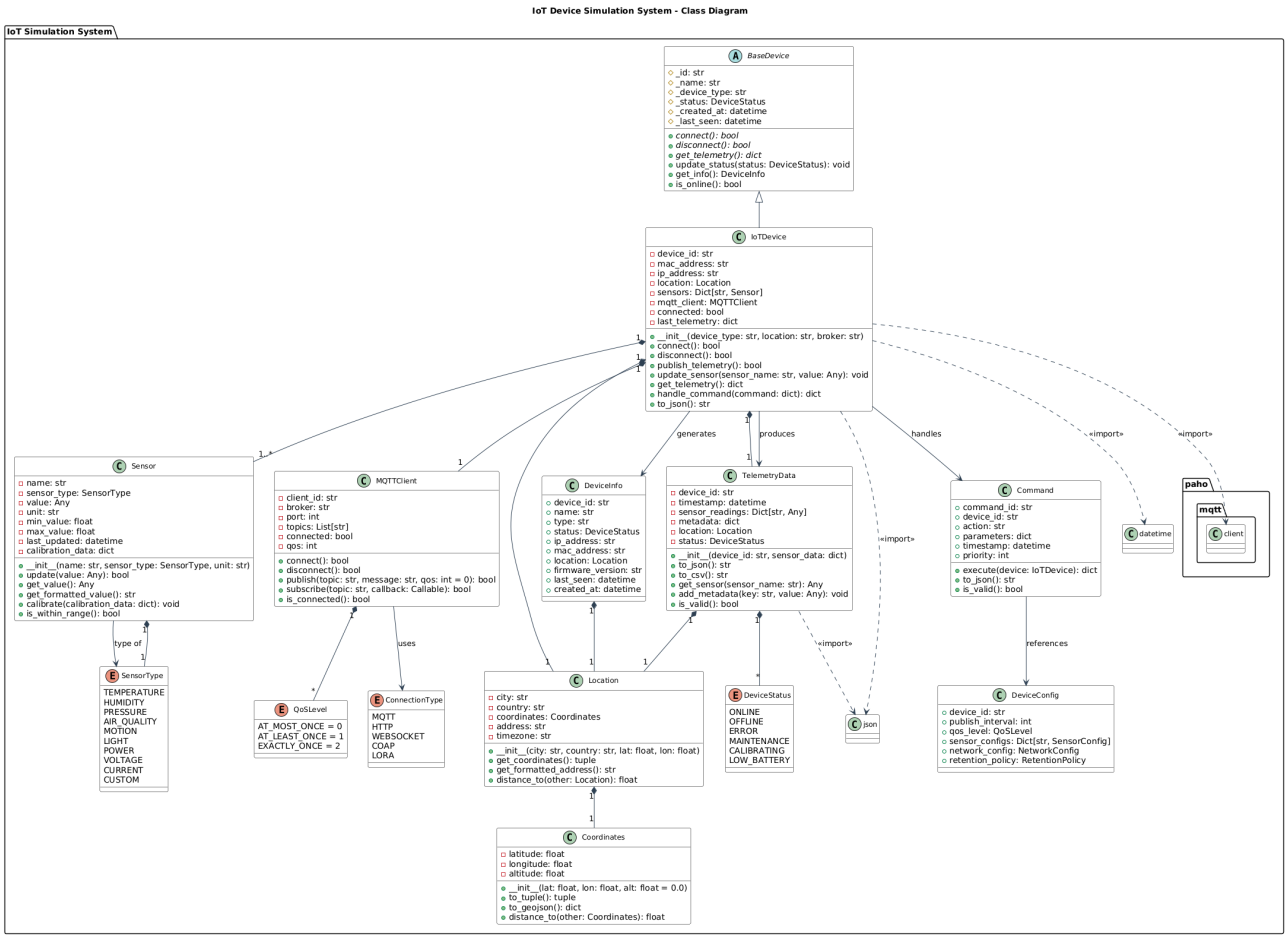

Fig 1.3 Iot Class Diagram

Steps

Step 1: Micro-services

- Signing:
  - Register/Login users.
  - Manage JWT-based authentication.
- Device Management:
  - Add/edit/delete/search devices.
  - Publish device events to RabbitMQ.
- Monitoring:
  - Listen to RabbitMQ for data ingestion.
  - Provide APIs for querying device data.
  - Implement Socket.IO for real-time updates.

Project structure

- Microservices
  - device-management
    - dal
    - business
    - models
    - controllers
    - config
    - helpers
    - Tests {rest client}
    - dockers
    - k8s
  - app.py
  - signing
  - monitoring
  - iot-devices {simulate hot}
  - end-devices {for end device}

Step 2: Build Micro-services

- **Signing Microservice:**
  - Implement user authentication in FastApi.
  - Use PostgreSQL for user data.
  - Use Redis for session caching.
- **Device Management Micro service:**
  - Implement device management logic in FastApi.
  - Store device details in PostgreSQL.
  - Publish device events to RabbitMQ.
- **Monitoring Microservice:**
  - Consume RabbitMQ messages to store IoT data in MongoDB.
  - Implement APIs to retrieve monitoring data.
  - Use Socket.IO for real-time updates.

Step 3: Containerization for testing

- Write Dockerfile for each backend (use tag for each image)micro service.
- Use **Docker Compose** locally to test the system before deploying to Kubernetes.

Step 5: Deployment in k8s

- Use kompose to convert compose file to deployment and services

Dev Librairies

- fastapi-mqtt, paho_mqtt : MQTT protocol {done}
- requests : to communicate using http from end-device to server
- sklearn : prediction (https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html) case of api open-meteo
- SQLAlchemy / pymongo pour le mapping objet database
- matplotlib for visualisation visualisation
- psutil : get information about cpu, disk usage and memory for end devices

Devops Tools

- git (fork my repository in git and invite your colleague)
- docker / docker compose or podman (containerisation)
- microk8s or K3s (k8s environment)
- sonarqube (code quality and security)