UNIVERSIDADE DE ÉVORA

**Sistemas Distribuídos**
2º trabalho
2025/2026

# Sistema de Monitorização Ambiental para a Universidade de Évora

## 1. Objetivo

O objetivo deste trabalho é desenvolver um sistema distribuído para monitorizar a temperatura e a humidade de diversas áreas da Universidade de Évora. O sistema deverá receber dados enviados por diferentes tipos de dispositivos IoT utilizando múltiplos protocolos de comunicação (MQTT, gRPC e REST), processá-los, armazená-los e disponibilizar funcionalidades para consulta e gestão.

---

## 2. Descrição da Tarefa

A Universidade de Évora, com os seus diversos edifícios e instalações, necessita de um sistema eficiente e automatizado para monitorizar e gerir a temperatura e a humidade de todos os seus espaços. Estas variáveis são cruciais para garantir o conforto de estudantes e funcionários, preservar equipamentos sensíveis (servidores, laboratórios) e cumprir normas de controlo ambiental em áreas críticas, como bibliotecas, arquivos históricos e laboratórios de investigação.

Este trabalho prático visa a implementação de um sistema distribuído que permitirá:

- **Monitorizar** temperatura e humidade em diferentes áreas da universidade utilizando três protocolos diferentes;
- **Registar, consultar, atualizar e eliminar** dispositivos IoT através de endpoints RESTful;
- **Receber métricas** de dispositivos através de três protocolos distintos:

    - **MQTT** para sensores IoT simples (baixo consumo energético)
    - **RPC** para dispositivos edge/gateways (alto desempenho)
    - **REST** para dispositivos com capacidade HTTP (simplicidade)
- **Consultar** métricas ambientais agregadas por sala, departamento, piso ou edifício, com opções de filtragem.

---

# 3. Requisitos do Sistema

## 3.1 Dispositivos IoT (Clientes que Enviam Dados)

O sistema deve suportar **três tipos de clientes** que enviam dados ao servidor, cada um utilizando um protocolo diferente:

### 3.1.1 Clientes MQTT (Sensores IoT Simples)

- Dispositivos IoT de baixo consumo instalados na universidade enviam dados através de um **broker MQTT (ActiveMQ)**.
- **Características**:
    - Comunicação assíncrona (pub/sub)
    - Ideal para sensores alimentados a bateria
    - Menor overhead de rede
- **Implementação**:
    - Desenvolver simuladores que, **ao iniciar, geram automaticamente dados sintéticos** (temperatura e humidade aleatórias dentro de intervalos realistas)
    - Publicam mensagens continuamente no broker MQTT
    - Podem simular múltiplos sensores numa única aplicação
- **Exemplos**: Sensores em salas de aula, biblioteca, laboratórios

### 3.1.2 Clientes gRPC (Dispositivos Edge/Gateways)

- Dispositivos mais robustos (gateways IoT, controladores industriais) enviam dados diretamente ao servidor via **gRPC**.
- **Características**:
    - Comunicação síncrona de alto desempenho
    - Protocolo binário eficiente
    - Contratos fortemente tipados (.proto)
    - Ideal para agregação de múltiplos sensores
- **Implementação**:
    - Desenvolver simuladores que, **ao iniciar, geram automaticamente dados sintéticos**
    - Chamam o serviço gRPC do servidor periodicamente.
    - Podem simular agregação de múltiplos sensores virtuais
    - Aguardam resposta do servidor (comunicação síncrona)

### 3.1.3 Clientes REST (Dispositivos com HTTP)

- Dispositivos simples com suporte HTTP/JSON enviam dados através de **endpoints REST**.
- **Características**:
    - Comunicação síncrona padrão
    - HTTP POST com payload JSON
    - Mais fácil de integrar e debugar
    - Universal e legível por humanos
- **Implementação**:
    - Desenvolver simuladores que, **ao iniciar, geram automaticamente dados sintéticos**
    - Fazem POST para o servidor periodicamente
    - Recebem resposta HTTP (200 OK, 400 Bad Request, etc.)
    - Podem incluir retry logic em caso de falha

#### 3.1.4 Formato dos Dados

Independentemente do protocolo utilizado, cada mensagem enviada por um dispositivo deverá conter:

- **Identificador único do dispositivo** (id)
- **Temperatura** registada
- **Humidade** registada
- **Timestamp** (data e hora do envio)

#### 3.1.5 Geração de Dados Sintéticos

Os simuladores devem gerar dados sintéticos realistas:

**Temperatura:**

- Intervalo: 15°C a 30°C
- Variação gradual (não saltos bruscos)
- Sugestão: usar número aleatório com alguma persistência temporal

**Humidade:**

- Intervalo: 30% a 80%
- Variação gradual
- Pode ter correlação com temperatura (opcional)

**Timestamp:**

- Data e hora atual do envio
- Formato ISO 8601 recomendado (ex: 2024-12-08T14:30:00Z)

## 3.2 Servidor Principal (Spring Boot)

O servidor centralizado deve:

### 3.2.1 Recepção Multi-Protocolo

Implementar **três camadas de ingestão de dados**:

1. **MQTT Subscriber**
    - Subscreve tópicos no broker MQTT
    - Recebe dados de sensores IoT simples
    - Processa mensagens assíncronas
2. **gRPC Service**
    - Implementa serviço gRPC definido em ficheiro .proto
    - Recebe dados de dispositivos edge de alto desempenho
3. **REST Controller (Ingestão)**
    - Endpoint: POST /api/metrics/ingest
    - Recebe dados em formato JSON
    - Valida e processa dados HTTP

### 3.2.2 Processamento e Validação dos Dados

- O servidor deve **validar** os dados recebidos de **qualquer protocolo**:
    - Caso o dispositivo não esteja registado no sistema, as métricas devem ser descartadas
    - Caso o dispositivo seja válido, as métricas devem ser processadas e armazenadas
- Utilizar **PostgreSQL** como base de dados
- Os dados devem ser organizados para permitir **consultas agregadas**

### 3.2.3 Gestão de Dispositivos IoT (API REST)

Implementar uma **API RESTful** para gestão de dispositivos:

- **POST /api/devices** - Criar: Registar dispositivos, associando-os a sala, departamento, piso e edifício
- **GET /api/devices** - Ler: Listar dispositivos registados
- **GET /api/devices/{id}** - Ler: Obter detalhes de um dispositivo
- **PUT /api/devices/{id}** - Atualizar: Alterar informações dos dispositivos
- **DELETE /api/devices/{id}** - Eliminar: Remover dispositivos do sistema

Cada dispositivo deve ter:

- ID único
- Tipo de protocolo utilizado (MQTT, gRPC ou REST)
- Localização (sala, departamento, piso, edifício)
- Estado (ativo/inativo)

### 3.2.4 Consultas de Métricas (API REST)

Implementar **endpoints REST** para consultar:

- **GET /api/metrics/average?level={sala|departamento|piso|edificio}&id={id}&from={date}&to={date}**
    - Retorna temperatura média e humidade média
    - Parâmetros de filtragem:
        - level: Nível de agregação (sala, departamento, piso, edifício)
        - id: ID da entidade (ex: sala_123, departamento_informatica)
        - from / to: Intervalo de datas (opcional, default: últimas 24h)
- **GET /api/metrics/raw?deviceId={id}&from={date}&to={date}**
    - Retorna métricas brutas de um dispositivo específico

## 3.3 Cliente de Administração (CLI)

Desenvolver uma **aplicação de linha de comandos** que permita:

### 3.3.1 Menu Principal

Menu com as seguintes opções

- Gestão de Dispositivos
- Consulta de Métricas
- Estatísticas do Sistema
- Sair

### 3.3.2 Gestão de Dispositivos

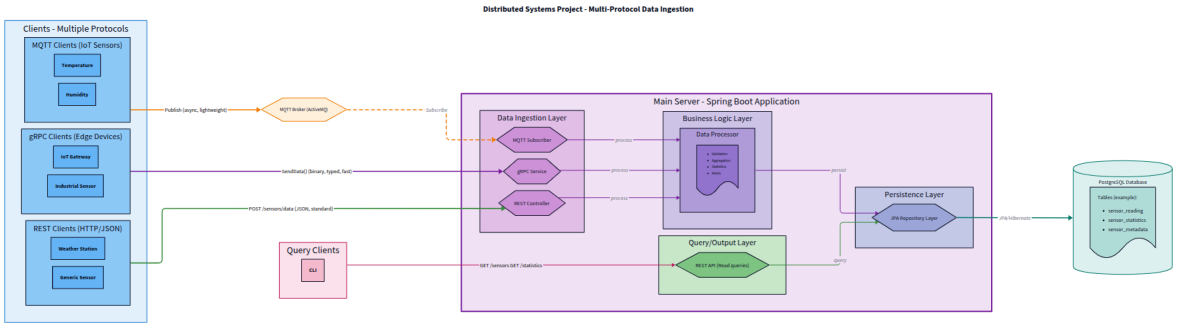
- Listar todos os dispositivos
- Adicionar novo dispositivo
- Atualizar dispositivo existente
- Remover dispositivo
- Visualizar detalhes (incluindo protocolo utilizado)

### 3.3.3 Consultas de Métricas

- Consultar por sala
- Consultar por departamento
- Consultar por piso
- Consultar por edifício
- Especificar intervalo de datas
- Visualizar dados em formato tabular

---

# 5. Arquitetura do Sistema




---

# 5. Tecnologias Obrigatórias

**Linguagem**: Java
**Framework**: Spring Boot
**Base de Dados**: PostgreSQL ou H2
**ORM**: JPA/Hibernate (criação automática de tabelas recomendada)
**MQTT**: ActiveMQ, Mosquitto ou outro broker
**gRPC**: gRPC Java com Protocol Buffers
**Build**: Maven ou Gradle

---

# 6. Entrega

- Enviar um ficheiro .zip no Moodle, contendo a pasta sd-t01-YYYYY-ZZZZZ (substituir YYYYY e ZZZZZ pelos números de aluno). Cada módulo é um projeto separado com a sua própria estrutura. Exemplo:

```
├── server/                    # Servidor Spring Boot
│   ├── src/
│   ├── pom.xml
│   └── application.properties
├── client-mqtt/               # Simulador MQTT
│   ├── src/
│   └── pom.xml
├── client-grpc/               # Simulador gRPC
│   ├── src/
```

```
│   └── pom.xml
├── client-rest/                # Simulador REST
│   ├── src/
│   └── pom.xml
├── admin-cli/                  # Cliente de administração
│   ├── src/
│   └── pom.xml
├── proto/                      # Ficheiros .proto do gRPC
│   └── metrics.proto
├── docker-compose.yml              # MQTT Broker + PostgreSQL
(opcional)
├── database/
│   └── schema.sql              # (se não usar JPA auto-create)
├── relatório.pdf
└── README.md
```

- A pasta deve conter o código-fonte, ficheiros de configuração, o script de execução das aplicações (pom ou gradle).
- Um relatório deve estar incluído com:
    - Identificação dos alunos.
    - **Justificação das escolhas feitas ao longo do trabalho** (ex.: estrutura das tabelas na base de dados e métodos de comunicação entre cliente e servidor)
    - Instruções detalhadas para a configuração da base de dados PostgreSQL, incluindo os passos necessários para criar a base de dados, tabelas, utilizadores e permissões necessárias para executar o sistema.
    - Observações sobre o desenvolvimento e eventuais desafios encontrados.
    - **Uma análise de performance que deve incluir uma comparação detalhada de performance entre os três protocolos, com base nos resultados obtidos no projeto.**

**Nota:** Os alunos podem optar por trabalhar **em pares, trabalho individual só com aprovação**.