Relatório do Segundo Trabalho de Sistemas Distribuídos

Sistema de Monitorização Ambiental para o Novo Hospital de Évora

Thawila Simbine - 49183

Jailson Lopes - 40699

Évora, Janeiro de 2025

1. Introdução

O objetivo do trabalho é desenvolver um sistema distribuído para monitorizar a temperatura e a humidade de diversas áreas do novo hospital de Évora. O sistema deverá receber dados enviados por dispositivos IoT instalados no edifício, processá-los, armazená-los e disponibilizar funcionalidades para consulta e gestão.

O trabalho consiste na implementação de um sistema distribuído que permitirá:

- Monitorizar temperatura e humidade em diferentes áreas do hospital;
- O registo, consulta, atualização e eliminação de dispositivos IoT disponibilizadas através de endpoints RESTful, permitindo o acesso utilizando operações padrão do protocolo HTTP, como GET, POST, PUT e DELETE.
- Os dispositivos IoT comunicam as métricas (temperatura, humidade e timestamp) para o servidor utilizando um broker MQTT, que será responsável por intermediar a transmissão dos dados.
- Consultar métricas ambientais agregadas por sala, serviço, piso ou edifício, com opções de filtragem.

2. Recursos usado para a implementação do servidor

Para a implementação do servidor, foram usados os seguintes recursos:

1. Framework: Java Spring Boot
2. Base de dados: PostgreSQL
3. Message Broker: MQTT

Dependências usadas na aplicação spring boot:

1. PostgreSQL Driver
2. Spring Boot JPA
3. Lombok
4. Spring Boot Web
5. Spring Boot Integration MQTT

3. Modelação do sistema

3.1. Modelação das entidades (ou tabelas da base de dados)

Na descrição do problema no enunciado foram apresentadas algumas opções de filtragem de dados (métricas) na base de dados, onde as filtragens podem ser feitas por sala, serviço, piso e edifício. Para que essas operações sejam possíveis, é necessário organizar os dados de modo a permitir tais consultas agregadas. Além disso, o enunciado aborda muito sobre os dispositivos IoT.

Nós achamos importante que os dados de todas as agregações requeridas no enunciado estejam salvas na base de dados, juntamente com os dados dos dispositivos IoT, por isso, primeiro modelamos cada uma dessas entidades para identificar os atributos/propriedades que cada uma delas possui, permitindo-nos assim saber como é que podemos salvá-las na base de dados.

1. **Características dos dispositivos IoT**:

- Cada dispositivo tem um ID único;
- Localiza-se numa sala dentro do hospital.

2. **Caracterização das Salas do hospital**:

- Têm um ID único cada sala;
- Têm um nome;
- Localizam-se num piso dentro de um dos edifícios do hospital;
- São prestados serviços nessas salas;

3. Caracterização dos Edifícios

- Têm um ID único cada edifício;
- Têm um nome;
- Têm pisos;

4. **Caracterização dos serviços**:

- Têm um ID único cada serviço;
- Têm um nome;

5. Caracterização das métricas:

- Cada métrica pertence a um dispositivos IoT;
- São compostas por temperatura, humidade e o timestamp em que foi capturada;

Depois da modelação das entidades passamos para a fase de codificação das mesmas. Todas as entidades criadas encontram-se no pacote *models*.

4. Organização do projeto

O projeto está organizado em 7 pacotes principais com os nomes:

1. ***models*** - pacote com as classes entidades (classes que mapeiam tabelas na base de dados). As classes entidades obtidas são:
    - Edificio
    - Sala
    - Servico
    - DispositivoIoT
    - Metricas
2. ***repositories*** - pacote com as interfaces repositório. Ou seja, interfaces onde são definidos métodos que permitem manipular dados na base de dados.
   Existe uma interface repositório para cada uma das classes entidades. Cada interface repositório define métodos para manipulação da tabela ligada a entidade a que ele representa.

3. ***services*** - pacote com classes de serviço. Tal como o nome sugere, trata-se de classes que implementam métodos que permitem realizar operações sobre o sistema. Todas as funcionalidades acessadas no sistema passam por estas classes.
   Existe uma classe de serviços para cada uma das classes entidades. Cada classe de serviço implementa métodos para acesso às operações relacionadas à entidade a que ele representa.
   Neste pacote tem um subpacote de nome ***dto*** onde estão localizadas as classes que permitem fazer passagem de dados da camada dos controllers para a camada dos serviços e vice-versa. A ideia é que os controllers não podem acessar as classes modelo diretamente. Os controllers recebem e enviam dados aos serviços por meio de DTOs (Data Transformer Objet).

4. ***controllers*** - pacote com as classes controladoras. É nessas classes onde são definidos os endpoints que permitem acessar cada recurso da aplicação. Essas classes são o elo de ligação entre os endpoints e os serviços definidos no pacote de serviços.
   Existe um controlador para cada uma das classes entidades. Cada controlador implementa endpoints para acesso aos recursos relacionados à entidade a que ele representa.

5. ***exceptions*** - pacote com as exceções personalizadas da aplicação. O nome dessas exceções foram definidas de modo que sejam altamente sugestivos, no sentido que logo que alguém ver a exceção logo saiba o que ela representa.
   As exceções definidas neste pacote são muito úteis para uso em combinação com ***Rest Controllers Advices*** no retorno de respostas com estado HTTP corretamente definidos, caso ocorra algum erro no meio de uma requisição ao servidor.
   Por exemplo a exceção ***EntidadeJaExistenteException*** sugere que um certo dado já existe registado na aplicação, o que remete a um estado de conflito. Quando suado esta exceção com os ***Rest Controllers Advices*** é possível definir que sempre esse exceção for lançada a aplicação retorne ao cliente o estado HTTP 409 (CONFLIT).

6. ***config*** - pacote com classes que definem configurações da aplicação ou operações que devem ser realizadas durante todo o período de execução da aplicação.
   A classe que implementa a rotina de leitura de mensagens enviadas pelos dispositivos IoT num broker MQTT por exemplo está definida neste pacote. Neste pacote também está definida uma classe que permite fazer a conversão de um objeto JSON é um objeto java (da classe MetricasDTO). Também está definida uma classe que permite fazer a inicialização da base de dados da aplicação, de modo que logo de início já tenha dados disponíveis para testes.
   Neste pacote também estão localizadas as classes ***Rest Controllers Advices*** dentro do pacote ***advice***.

7. ***utils*** - pacote com classes utilitárias. Neste pacote está definida somente uma classe.
   A classe aqui definida, permite executar uma rotina (função) que possivelmente podem lançar uma exceção como se estive numa caixa de areia, no sentido que caso a rotina lance uma exceção, a exceção é tratada lá dentro de modo que não se propague e afete as rotinas que seguem.

5. Simulador de dispositivo IoT

O simulador de dispositivo IoT por nós implementado é muito simples. Usando a biblioteca PahoClient, primeiro o simulador cria uma conexão com o message broker de url tcp://broker.mqttdashboard.com:1883. Em cada conexão é usado um clientId diferente gerado usando a classe ***UUID*** com o método randomUUI() para criar IDs aleatórios.

- Quando a conexão é estabelecida, é iniciado um loop infinito que em cada iteração:
    - Gera uma mensagem com os atributos ID do dispositivo IoT, temperatura e humidade aleatórios, onde:
        - O ID do dispositivo IoT gerado é um Long valor compreendido na faixa entre os 1 e 50
        - A temperatura gerada é um Double com valor compreendido na faixa entre 5 e 45
        - A humidade gerada é um Double com valor compreendido na faixa entre 1 à 100 (1% a 100%).
    - Depois publica a mensagem no tópico

O projeto do simulador tem o nome de ***SimuladorDispositivoIoT***

6. ClienteAdministracao

O Cliente Administracao por nós implementado reutilizando a logica do primeira trabalho

O projeto do cliente administracao tem o nome de **ClienteAdministracao**

Conclusão

O que funciona:

1. O Servidor
2. O Simulador de dispositivo IoT
3. A comunicação entre o servidor e o simulador de dispositivo IoT
4. O filtro que permite salvar apenas mensagens com ID de dispositivo IoT registados.
5. Endpoints que permitem REGISTAR, CONSULTAR, EDITAR e ELIMINAR dispositivos IoT.
6. Endpoints que permitem consultar métricas por dispositivo IoT.

O que não funciona ainda:

- Alguns tipos de consulta de métricas
- Autenticação
- Cliente de administração