Sistemas Distribuídos
Trabalho 1
Sistema de acompanhamento de vacinação

Luís Maurício 37722
Fábio Macarrão 41895

1 Introdução

Para este primeiro trabalho prático da unidade curricular de Sistemas Distribuídos era pedido que se desenvolvesse duas aplicações:

- **Servidor** - Encarregue de manter a persistencia de dados e dar resposta às funcionalidades implementadas no Cliente
- **Cliente** - Aplicação que implementará todas as funcionalidades pedidas no enunciado

Utilizámos o Spring Data JPA para manter a persistência de dados numa base de dados PostgreSQL e, como forma de comunicação entre o Cliente e o Servidor, optámos por utilizar serviços REST.

Para gestão de dependências escolhemos o Gradle.

O Cliente corre apenas na linha de comandos como um programa Java sem utilizar framework, enquanto o Servidor utiliza a framework Spring Boot.

2 Modelos

Os modelos utilizados são duplicados, havendo uma cópia no Servidor e outra no Cliente. A diferença é que os modelos presentes no Servidor têm as anotações para persistência do Spring Data JPA.

2.1 Centro

Este modelo representa um centro de vacinação.

Variáveis:

- *Long* **id** - identificador único, utilizado como chave primária na base de dados e como identificador nos pedidos que o Cliente faz em que necessita indicar o centro.
- *String* **cidade** - nome da cidade onde o centro se encontra.
- *List<Vacinacao>* **listaMarcacoes** - Lista para guardar todas as vacinação agendadas e efetuadas no centro em questão.

Métodos:

Para além de construtor, getters e setters, implementámos um método que devolve a contagem apenas de vacinações agendadas para poder mostrar o comprimento da fila de espera.

2.2 Vacinacao

Este modelo representa uma incrição para vacinação e também vacinações já realizadas.

Variáveis:

- *Long* **id** - identificador único, utilizado como chave primária na base de dados e como identificador nos pedidos que o Cliente faz em que necessita indicar o código de inscrição.
- *String* **nome** - nome do cidadão que fez o pedido
- *String* **tipo** - tipo de vacina administrada. Esta variável só é preenchida após a vacinação em causa ser realizada
- *int* **idade** - idade do cidadão
- *Date* **dataAdministracao** - data em que o cidadão foi vacinado. Esta variável só é preenchida após a vacinação em causa ser realizada
- *boolean* **efeitos** - variável que indica se o cidadão sofreu efeitos secundários os não. Esta variável só é preenchida após a vacinação em causa ser realizada
- *boolean* **administrada** - variável que indica se a vacina está só agendada ou já realizada. Esta variável só é preenchida após a vacinação em causa ser realizada
- *char* **genero** - caracter para identificar o género do cidadão

Métodos

Este modelo só tem mesmo os construtores, getters e setters.

3 Servidor

O Servidor é composto por 7 classes, das quais 2 são controllers, 2 repositórios, 2 modelos e 1 para o método principal com a anotação @SpringBootApplication.

3.1 Controllers

Os Controllers são classes responsáveis por receber e enviar pedidos web.

3.1.1 CentroController

Esta classe utiliza a anotação @RestController do Spring.

Variáveis:

- *CentroRepo* **centroRepo** - instância do repositório referente aos centros

Métodos

- *ResponseEntity<List<Centro>>* **all()**:
  Retorna todos os centros no formato de lista dentro de um objeto ResponseEntity. Este método é executado através de um pedido GET para o endereço do servidor + /centros
- *ResponseEntity<Centro>* **one(Long id)**:
  Recebe um id através do url e responde com um objeto Centro referente ao id dentro de um ResponseEntity, caso não exista, o objeto da resposta fica vazio. Este método é executado através de um pedido GET para o endereço do servidor + /centros/ + id

- *Centro* **newCentro(Centro centro)**:
  Recebe um objeto Centro e guarda-o utilizando o repositório, responde com o Centro que guardou (só ao guardar é que lhe é atribuído um id). Este método é executado através de um pedido POST para o endereço do servidor + /centros , com o objeto a guardar no corpo do pedido.

- *Centro* **replaceCentro(Centro newCentro, Long id)**:
  Recebe um objeto Centro e um identificador e atualiza o centro que tem esse id substituindo-o pelo que é recebido no corpo do pedido. No caso de não existir nenhum Centro com o id pedido, é criado um novo. Este método é executado através de um pedido PUT para o endereço do servidor + /centros/ + id.

3.1.2 VacinacaoController

Esta classe utiliza a anotação @RestController do Spring.

Variáveis:

- *VacinacaoRepo* **vacinacaoRepo** - instância do repositório referente às vacinações

Métodos

- *ResponseEntity<List<Vacinacao>>* **all()**:
  Retorna todas vacinações no formato de lista dentro de um objeto ResponseEntity. Este método é executado através de um pedido GET para o endereço do servidor + /vacinacoes.

- *ResponseEntity<Vacinacao>* **one(Long id)**:
  Recebe um id através do url e responde com um objeto Vacinacao referente ao id dentro de um ResponseEntity, caso não exista, o objeto da resposta fica vazio. Este método é executado através de um pedido GET para o endereço do servidor + /vacinacoes/ + id.

- *Vacinacao* **newVacinacao(Vacinacao vacinacao)**:
  Recebe um objeto Vacinacao e guarda-o utilizando o repositório, responde com a Vacinacao que guardou (só ao guardar é que lhe é atribuído um id). Este método é executado através de um pedido POST para o endereço do servidor + /vacinacoes , com o objeto a guardar no corpo do pedido.

- *Vacinacao* **replaceVacinacao(Vacinacao newVacinacao, Long id)**:
  Recebe um objeto Vacinacao e um identificador e atualiza a Vacinacao que tem esse id substituindo-a pela que é recebida no corpo do pedido. No caso de não existir nenhuma Vacinacao com o id pedido, é criado um novo. Este método é executado através de um pedido PUT para o endereço do servidor + /vacinacoes/ + id.

3.2 Repositórios

3.2.1 CentroRepo

Repositório CRUD para aceder à tabela dos centros. Tem apenas dois métodos: Centro findById(long id) e List<Centro> findAll().

3.2.2 VacinacaoRepo

Repositório CRUD para aceder à tabela das vacinações. Tem apenas dois métodos: Vacinacao findById(long id) e List<Vacinacao> findAll().

3.3 application.properties

Este ficheiro guarda propriedades necessárias para correr o Servidor. Tem apenas configurações relativas à base de dados.

4 Cliente

A aplicação Cliente tem apenas os 2 modelos já referidos e uma classe ClientApplication que corresponde à aplicação de linha de comandos que pode ser utilizada para comunicar com o servidor.

4.1 ClientApplication

Variáveis

- *Properties* **properties** - variável que guarda as propriedades.
- *RestTemplate* **rt** - variável utilizada para fazer pedidos GET, POST, PUT e DELETE.

Métodos

- *void* **main(String[] args**:
  Inicializa uma instância do ClientApplication e chama o método menu deste.
- *void* **menu()**:
  Método que chama os métodos corretos para as ações escolhidas pelo cliente.
- *void* **consultaCentros()**:
  Método que faz o pedido GET para listar os centros existentes e escreve a lista para o stdout.
- *void* **filaEspera(Long id)**:
  Método que faz o pedido GET para receber o centro com o id do argumento e chama o método countFila() para poder escrever o número de pessoas em espera para ser vacinadas nesse centro.
- *void* **inscrever(Long id, String data, char gen, int idade)**:
  Cria uma instância de Vacinacao que envia através de um pedido POST ao servidor para este a guardar na lista de espera do centro identificado pelo id dos argumentos.

- *void* **realizar(Long id, String data, String tipo)**:
  Recebe o id da vacinação, a data em que foi realizada (no formato dd-mm-aaaa) e o tipo de vacina utilizada. Faz um pedido GET ao servidor para receber o objeto Vacinacao com o id referido. Se essa vacina ainda estiver em espera regista a data de administração, o tipo e torna o campo administrada a true e envia um pedido PUT para esta ser atualizada.

- *void* **efeitos(Long id)**:
  Faz um pedido GET ao servidor para receber a Vacinacao com o id. Se esta já tiver sido administrada coloca o campo efeitos a true e envia o objeto através de um pedido PUT para que o servidor guarde a nova versão.

- *void* **administradasPorTipo()**:
  Recebe a lista de todas as vacinações após fazer um pedido GET ao servidor. Corre a lista para descobrir quantos tipos existem e depois volta a correr a lista por cada tipo encontrado para fazer a contagem de quantas vacinas foram administradas desse tipo.

- *void* **efeitosPorTipo()**:
  Semelhante ao método administradasPorTipo mas procura por vacinações que tenham o campo efeitos true.

- *int* **intInput(String campo)**:
  Método que trata de todos os inputs que sejam do tipo inteiro. Dependendo da string passada como argumento avisa o utilizador que informação está a ser pedida e repete-se até que o input seja o esperado.

- *String* **strInput(String campo)**:
  Método que trata de todos os inputs que sejam do tipo String. Dependendo da string passada como argumento faz a validação do input recorrendo a expressões regulares.

4.2 application.properties

Ficheiro de configuração, contém apenas a propriedade url que indica o endereço do servidor.