Sistemas Distribuídos
Trabalho 2
Sistema de acompanhamento de vacinação

Luís Maurício 37722
Fábio Macarrão 41895

1 Introdução

Para este trabalho era pedido que se desenvolvesse 3 aplicações:

- **Cidadão** - Aplicação que pode ser utilizada por qualquer cidadão para agendar a vacinação
- **Centro** - Aplicação que representa um centro municipal.
- **DGS** - Módulo central que guarda os dados e responde a pedidos dos centros e cidadãos

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

As aplicações cidadão e centro correm apenas na linha de comandos como um programa Java sem utilizarframework, enquanto o módulo da DGS utiliza a framework Spring Boot e corre com um CommandLineRunner desta.

Para gestão de dependências escolhemos o Gradle.

2 Modelos

Criámos 4 modelos no módulo da DGS, dos quais 3 também existem nos outros módulos (Centro, Vacinacao e Notificacao). A diferença é que os modelos presentes no módulo da DGS 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.
- *int* **capacidade** - capacidade diária de vacinação.
- *List<Vacinacao>* **listaMarcacoes** - Lista para guardar todas as vacinação agendadas e efetuadas no centro em questão.
- *List<VaxPorDia>* **vaxPorDia** - Lista para guardar número de vacinas alocadas para um centro num dia especifico. Esta lista só está presente no módulo da DGS.

Métodos:

Este modelo apenas tem construtores, getters e setters no módulo da DGS e do cidadão, já na aplicação do centro implementámos um método que retorna uma lista de vacinações agendadas para um dia específico, ordenada por idades descendente.

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 cidadão/centro faz em que necessita indicar o código de inscrição.
- *String* **nome** - nome do cidadão que fez o pedido.
- *String* **email** - email 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* **dataPreferida** - data que o cidadão escolheu ao autoagendar.
- *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. Esta variável só é preenchida após a vacinação em causa ser realizada.
- *boolean* **administrada** - variável que indica se a vacinação está agendada ou já realizada. Esta variável só é preenchida após a vacinação em causa ser realizada.
- *boolean* **confirmada** - variável que indica se a vacina já foi confirmada. Acontece depois da DGS distribuir vacinas a nível nacional para um dia e o centro pedir a quantidade que depois distibui pelos cidadãos inscritos.
- *List<Notificacao>* **notificacaoList** - lista que guarda as notificações para poderem ser mostradas mais tarde ao cidadão.
- *Long* **centroId** - id do centro onde a vacinação foi agendada/realizada.

Métodos

Este modelo tem os construtores, getters, setters e 3 métodos especializados:

- *boolean* **isUnread()** - indica se existem notificações por ler associadas à vacinação. Apenas utilizado na aplicação do cidadão.
- *List<Notificacao>* **unread()** - retorna uma lista com todas as notificações não lidas associadas à vacinação. Apenas utilizado na aplicação do cidadão.
- *void* **addNotificacao(Notificacao n)** - adiciona a notificação n à lista de notificações deste registo de vacinação. Apenas utilizado na aplicação do centro.

2.3 Notificacao

Este modelo representa uma notificação associada a uma vacinação para chegar ao cidadão.

Variáveis:

- *Long* **id** - identificador único, utilizado como chave primária na base de dados.
- *Long* **vacinacaoId** - identificador da vacinação a que a notificação está associada.
- *String* **titulo** - titulo da notificação.
- *String* **descricao** - texto a descrever a situação.
- *boolean* **lida** - variável para indicar que notificações já foram lidas.

Métodos

Este modelo tem apenas construtores, getters e setters.

2.4 VaxPorDia

Este modelo serve para guardar o número de vacinas que cada centro irá receber num determinado dia, apenas está implementado no módulo da DGS.

Variáveis:

- *Long* **id** - identificador único, utilizado como chave primária na base de dados.
- *Long* **centroId** - identificador do centro a que este objeto pertence.
- *Date* **data** - dia a que este objeto se refere.
- *int* **numVacinas** - número de vacinas disponíveis na data para o centro.

Métodos

Este modelo tem construtores, getters, setters e um método extra que incrementa em uma unidade o numVacinas.

3 DGS

O módulo da DGS é o mais completo e tem 4 controllers, 4 modelos, 4 repositórios, 2 exceções, 2 advices, a classe correspondente ao SpringBootApplication e ainda uma classe que implementa um CommandLineRunner.

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.

3.1.3 NotificacaoController

Esta classe utiliza a anotação @RestController do Spring.

Variáveis:

- *NotificacaoRepo* **notificacaoRepo** - instância do repositório referente às notificações.

Métodos

- *ResponseEntity<Notificacao>* **getNotificacao(Long id)**:
  Recebe um id através do url e responde com um objeto Notificacao 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 + /notificacoes/ + id.

- *Notificacao* **newNotificacao(Notificacao notificacao)**:
  Recebe um objeto Notificacao e guarda-o utilizando o repositório, responde com a Notificacao 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 + /notificacoes , com o objeto a guardar no corpo do pedido.

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

3.1.4 VaxPorDiaController

Esta classe utiliza a anotação @RestController do Spring.

Variáveis:

- *VaxPorDiaRepo* **vaxPorDiaRepo** - instância do repositório referente aos números de vacinas atribuídas por dia.

Métodos

- *ResponseEntity<List< VaxPorDia>>* **all()**:
  Retorna todos os registos de número de vacinas atribuídas no formato de lista dentro de um objeto ResponseEntity. Este método é executado através de um pedido GET para o endereço do servidor + /vaxpordia.

- *Integer* **numeroVacinas(String date, String centroId)**:
  Recebe uma data e um identificador do centro e procura no repositório por registos que correspondam a esse dia e esse centro. Retorna o número de vacinas que o centro irá receber nesse dia. Este método é executado através de um pedido GET para o endereço do servidor + /vaxpordia/ + date/centroId.

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 os seguintes métodos:

- Vacinacao findById(long id)
- List<Vacinacao> findAll()
- List<acinacao> findByDataAdministracaoAndTipo(Date date, String tipo)
- List<Vacinacao> findByDataPreferidaAndAdministradaOrderByIdade(Date date, boolean administrada)

3.2.3 NotificacaoRepo

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

3.2.4 VaxPorDiaRepo

Repositório CRUD para aceder à tabela vaxpordia. Tem os seguintes métodos: VaxPorDia findByCentroIdAndData(Long centroId, Date date), VaxPorDia findByData(Date date) e List<VaxPorDia> findAll()

3.3 application.properties

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

4 Cidadão

A aplicação Cidadão tem apenas os 3 modelos já referidos e uma classe CidadaoApplication que corresponde à aplicação de linha de comandos que pode ser utilizada para comunicar com o módulo da DGS.

4.1 CidadaoApplication

Variáveis

- *Properties* **properties** - variável que guarda as propriedades.
- *Vacinacao* **vacinacao** - variável utilizada para memorizar que marcação está a ser acedida por esta instância da aplicação.

Métodos

- *void* **main(String[] args**:
  Inicializa uma instância do CidadaoApplication e chama o método menu deste.
- *void* **menu()**:
  Método que chama os métodos corretos para as ações escolhidas pelo cidadão.
- *void* **saveNotificacao(Notificacao notificacao)**:
  Método que faz um pedido PUT ao servidor para guardar uma notificação.
- *Vacinacao* **getVacinacao(Long id))**:
  Método que faz um pedido GET ao servidor para receber a vacinação com id como identificador.
- *Centro[]* **consultaCentros()**:
  Método que faz o pedido GET para listar os centros existentes e escreve a lista para o stdout.
- *Vacinacao* **autoagendar(Long centroId, String nome, int idade, String email, String data)**:
  Cria um objeto Vacinacao e envia-o para o servidor através de um pedido POST para ser guardado na base de dados.
- *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.

5 Centro

A aplicação Centro tem apenas os 3 modelos já referidos e uma classe CentroApplication que corresponde à aplicação de linha de comandos que pode ser utilizada para comunicar com o módulo da DGS.

5.1 CentroApplication

Variáveis

- *Properties* **properties** - variável que guarda as propriedades.
- *Vacinacao* **vacinacao** - variável utilizada para memorizar que centro está a aceder por esta instância da aplicação.

Métodos

- *void* **main(String[] args**:
  Inicializa uma instância do CentroApplication e chama o método menu deste.
- *void* **menu()**:
  Método que chama os métodos corretos para as ações escolhidas pelo centro.

- *Centro* **getCentro(Long id))**:
  Método que faz um pedido GET ao servidor para receber o Centro com id como identificador.

- *Centro* **newCentro(int cap, String cidade)**:
  Método que faz um pedido POST ao servidor para guardar um novo centro.

- *void* **registarVacinaAgendada(Long id, String tipo, String data)**:
  Método que só trabalha localmente. A partir da variável global centro procura na lista de marcações por uma que tenha o id correto, no caso de encontrar e essa estar ainda em espera é administrada e é lhe atribuída um tipo. Fica apenas guardada localmente e é comunicada mais tarde ao módulo da DGS para guardar na base de dados.

- *void* **receberNVacinas(String data)**:
  Utiliza um pedido GET para saber quantas vacinas irão ser disponibilizadas na data para este centro. Distribui o número de vacinas recebidas correndo a lista de marcações por administrar e ordenadas por idade de forma a que os mais velhos fiquem com a marcação confirmada. Cria Notificações para avisar cada marcação que foi confirmada/cancelada e envia-as para a base de dados através de um pedido POST.

- *void* **saveCentro()**:
  Envia o centro que está a ser utilizado para a DGS através de um pedido PUT. Serve para comunicar a lista de vacinas realizadas ao fim do dia bem como qualquer alteração que aconteça no centro.

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

- *String* **inputStr(int campo)**:
  Método que trata de todos os inputs que sejam do tipo String. Dependendo do inteiro passado como argumento faz a validação do input recorrendo a expressões regulares.

5.2 application.properties

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