Universidade de Évora

Curso de Engenharia Informática

Sistemas Distribuidos

Relatório do Trabalho da componente prática

2020/2021

Controlo de Vacinação

Diogo Solipa-nº43071

Leonardo Catarro-nº43025

Introdução

O seu trabalho é implementar aplicação servidor (que terá os dados, em armazenamento persistente) e a aplicação cliente com as funcionalidades:

• consulta de centros de vacinação

• consulta do comprimento da fila de espera num centro

• inscrição para vacinação num dos centros (indicando o nome, género e idade)

• registar a realização de vacinação prevista para a inscrição com o ćodigo X (removendo o cidadão da fila de espera do centro em que se encontrava), ficando registada a data e tipo de vacina

• reportar existência de efeitos secundários para um cidadão antes vacinado com o código C

• listar nº total de vacinados por tipo de vacina e nº de casos com efeitos secundários por tipo de vacina.

Neste trabalho era pedida uma aplicação cliente-servidor, através de uma solução de middleware apresentada nas aulas. Então nós usamos JavaRMI para essa solução!

Estruturação do Código

Para este trabalho foi implementado uma interface remota VaccineCenter, com a sua implementação na classe VaccineCenterImpl e representa todas as funcionalidades implementadas para este trabalho:

```java
package sd;


import java.util.LinkedList;
import java.util.List;

public interface VaccineCenter extends java.rmi.Remote{

    LinkedList<VacCenterEntity> listAllVaccineCenters() throws Exception;

    Integer checkVacCenterQueueSize(Integer centerId) throws Exception;

    Integer addPersonToVaccinationApplication(String name, Integer age, Character gender, Integer centerId) throws Exception;

    Integer reportSecundaryEffects(Integer vaccinationCode, String reportEffects) throws Exception;

    Integer registryFinishedVaccination(Integer vaccinationCode, String vaccineType) throws Exception;

    Integer totalVaccinations(String vaccine) throws Exception;

    Integer totalSecEffects(String vaccine) throws Exception;

    List<String> getAllVaccines() throws Exception;
}
```

Para auxilio em alguns metódos implementados, implementamos 3 entidades/modelos:

- PersonEntity

- VacCenterEntity

- VacCenterQueueEntity

Esta entidades tem a estrutura de cada uma das tabelas da base de dados como forma de facilitir a apresentação dos dados no lado do cliente e no lado do servidor.

Tabelas na Base de Dados

Para ajudar na persistência de dados, para este trabalho era nos fornecida uma BD em postgres no site da universidade.

Nesta db inserimos 3 tabelas - person, vaccenter, center_queue, com a seguinte estrutura:

person:

```
        create table person (
    person_id SERIAL,
    name VARCHAR(100),
    age INT,
    gender CHAR(1),
    vaccinated BOOL,
    sec_effects VARCHAR(200),
    vaccine_type VARCHAR(100),
    vaccine_code serial,
    vaccination_date DATE,
    primary KEY(person_id)
);
```

vaccenter:

```
        create table vaccenter(
    center_id serial,
    name VARCHAR(100),
    center_location VARCHAR(100),
    primary KEY(center_id)
);
```

center_queue:

```
        create table center_queue (
    person_id  INT,
    center_id  INT,
    center_name VARCHAR(100),
    person_name VARCHAR(100),
    person_age INT
);
```

Esta tabelas representam, respetivamente, os utilizadores da aplicação, os centros de vacinação existentes(inserido com código java na primeira vez que é corrido o código do servidor) e a fila de espera dos centros de vacinação.

Servidor e Cliente

Como foi acima mencionado a comunicação entre o servidor e cliente foi feita através de JavaRMI usando o objeto remoto VaccineCenter.

A implementação do servidor foi quase concebida na totalidade na aula prática, porém levou algumas alterações de modo a ser ajustada ao trabalho em questão.

Do mesmo modo, o Cliente sofreu também as alterações necessárias, incluído a “construção” de um método para mostrar o menu(em terminal) do trabalho.

Funcionalidades

public LinkedList<VacCenterEntity> listAllVaccineCenters() throws Exception : este método retona uma Lista com todos os centros de vacinação existentes.

Este executa uma query sql para conseguir “Puxar” da bd os dados necessários.

public Integer checkVacCenterQueueSize(Integer centerId) : Apartir do argumento deste método e recorrendo a uma query faz um count por personId quando o Identificador do centro for igual ao passado em centerId.

public Integer addPersonToVaccinationApplication(String name, Integer age, Character gender, Integer centerId) : no lado do cliente usando um objeto da classe BufferedReader é pedida toda a informação que é passada com argumentos. Depois, é executada uma query que insere uma nova linha da na tabela center_queue e na tabela person

public Integer reportSecundaryEffects(Integer vaccinationCode, String reportEffects) : a partir do código de vacinação, único para cada pessoa e igual ao person_id(columna da tabela person) é pedido ao utilizador que insira os efeitos secundários após ser aplicada a vacina. Entretanto, é dado update a partir de código sql na linha respetiva na tabela person.

public Integer registryFinishedVaccination(Integer vaccinationCode, String vaccineType) : este método dá por aplicada a vacina, ou seja, através do vaccinationCode e da vaccineType(vacina tomada) inserida no lado do cliente, o metódo acede à tabela pessoa alterando o campo vaccinated de false para true e registando a data da conclusão da vacinação e de seguida remove a pessoa da tabela center_queue(retira-a da fila de espera).

public Integer totalVaccinations(String vaccine)

public Integer totalSecEffects(String vaccine)

public List<String> getAllVaccines()

Estes ultimos 3 métodos são usados como um só, respondendo ao requisito de listar o número total de vacinados por tipo de vacina e número total de casos com efeitos secundários por tipo de vacina

Problemas Encontrados e Observações

No desenvolvimento deste trabalho o maior problema encontrado terá sido possivelmente na construção de queries, porém com alguma pesquisa facilmente todos os problemas foram corrigidos!

Em termos gerais, conseguimos aplicar os conhecimentos aprendidos em Sistemas Distruibidos, conseguindo então ter todos os requisitos implementados e a funcionar corretamente.

Como Compilar e Correr o Programa

→ Servidor e Cliente no mesmo computador

1) Abrir o terminal no diretório do projeto:

2) Compilar as classes java:

javac -d classes/ src/main/java/sd/*.java

3) Correr o Servidor

java -cp postgresql-42.2.20.jar:classes sd.Server “host” “port”

4) Correr o cliente

java -cp postgresql-42.2.20.jar:classes sd.Cliente “host” “port”

Nota: “host” e “port” são argumentos necessários à execução do programa!

→Servidor e cliente em computadores diferentes

1) Abrir terminal do diretório do projeto

2) No Pc server:

executar 2) e 3) acima mencionados

3) No pc cliente:

Correr .jar resultante do comando mvn clean install com o comando:

java -cp (nomeFicheiro).jar sd.(Classname) (list of arguments)