Universidade de Évora

Relatório do Trabalho Prático
Sistemas Distribuídos

Isabel Nunes (nº43091) Ludgero Teixeira (nº41348)

Docente: Professor José Saias

Abril de 2021

1 Introdução

O 1º trabalho desta disciplina consiste na implementação de um Sistema de acompanhamento de vacinação.

A aplicação servidor e a aplicação cliente, teram as seguintes funcionalidades:

1. Consulta de centros de vacinação.
2. Consulta do comprimento da fila de espera num centro.
3. Inscrição para vacinação num dos centros.
4. Registar a realização de vacinação.
5. Reportar existência de efeitos secundários.

2 Persistência dos dados

Para a persistência de dados foi criada uma base de dados Postgres onde foram criadas as seguintes tabelas:

2.1 Fila de Espera

Esta tabela tem as seguintes relações:

- **NomeCentro**: nome de todos os locais de vacinação.
- **NomePessoa**: nome das pessoas que se encontram na fila de espera.
- **Idade**: idade das pessoas que se encontram na fila de espera.
- **Genero**: género das pessoas que se encontram na fila de espera.

- **CodicoX**: código único atribuído ás pessoas que se encontram na fila, é a chave primária desta tabela, sendo este é um código único. Como este atributo não se repete, será mais fácil registar a pessoa como vacinada.

2.2 Centros

Esta tabela tem as seguintes relações:

- **NomeCentro**: nome de todos os locais de vacinação, esta relação é uma chave estrangeira que vem da tabela **Fila de espera**.

2.3 Vacinados

Esta tabela tem as seguintes relações:

- **TipoVacina**: nome da vacina.
- **DataVacinado**: data em que a vacina foi administrada a uma determinada pessoa.
- **EfeitosSecundarios**: a existência ou não de um possível efeito secundário após a vacinação.
- **CodicoC**: código único atribuído ás pessoas após a sua vacinação. É a chave primária desta tabela sendo este é um código único. Como este atributo não se repete, será mais fácil registar a pessoa como tendo efeitos secundários.

3 Solução *Middleware*

Para a implementação desta aplicação distribuída houve a necessidade de vários processos distintos e simultâneos executarem operações noutros processos. Desta maneira, foi pensada uma solução *Middleware* para manter a abstração do sistema, escondendo a heterogeneidade dos seus componentes. O modelo Java RMI (*Remote Method Invocation*) resolveu estes problemas, baseando na invocação remota de métodos doutro objecto num processo diferente.

A abstração RMI oferece ainda outras garantias como o reenvio do pedido para o servidor até a resposta chegar ou a detecção de problemas no servidor, a filtragem de duplicados e a retransmissão de resultados.

Em RMI, o servidor fica concorrente, ou seja, vários clientes podem ligar-se simultâneamente.

4 Classes Implementadas

Este Sistema Distribuído tem as seguintes classes:

- **ProgresConector.java**: Tendo o objeto remoto e a interface remota, esta classe conecta os mesmos com a base de dados criada localmente.
- **Vacinacao.java**: Interface remota onde estão declarados os métodos remotos.
- **VacinacaoImpl**: Classe do objeto remoto onde estão implementadas as funcionalidades do serviço.
- **VacinacaoServer**: Cria uma instância do objeto remoto e regista-o no *binder*.

- **VacinacaoClient.java**: Esta classe inicia com um *lookup* ao objeto no *binder* ficando com a referência remota associada ao *proxy*, desta maneira ele pode invocar os métodos associados ao objeto, ou seja as funcionalidades do serviço.

5 Comandos para execução

Antes de executar qualquer comando é necessário ter uma Base de Dados local em *Postgres* onde se deve correr o *script* BD.sql, encontrado na pasta base, responsável pela criação das tabelas.

Para uma primeira execução do programa deve-se preencher no ficheiro **config.propreties** as respetivas credencias da base de dados criada (nome, utilizador e password), o *host* e porto utilizado para correr o java RMI.

A ordem pela qual os comandos devem ser executados é:

1. *make rmi*

   Compila os ficheiros e executa o serviço no porto presente no ficheiro **config.properties**.

2. *make server*

   Corre o servidor que vai ler os dados necessários para ativar o *binder* no ficheiro **config.properties**.

3. *make client*

   Corre o cliente que vai ler os dados necessários do ficheiro **config.properties** para criar o objeto remoto e ligá-lo ao *binder*. Este 3º comando deve ser corrido por cada cliente que se queira executar.

Cada um destes comandos deve ser executado num terminal diferente. Todas as operações deste programa foram testadas num ambiente *UNIX Linux Ubuntu* e em *MacOs*.

6 Conclusão

Por fim, considerámos que a realização deste trabalho foi positiva, pois foi possível metermos em prática o conhecimento teórico até ao momento abordado na unidade curricular.