2º trabalho - Room Rent

Licenciatura em Eng. Informática

Sistemas Distribuídos

UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Helder Godinho 42741
Guilherme Grilo 48921

Docente: José Saias

1 Introdução

De forma a tornar o sistema independente de uma só tecnologia/plataforma, isto é, possibilitar que um cliente ou utilizador se consiga conectar ao sistema, independentemente da linguagem em que se encontra implementado, foi necessário recorrer a uma API REST. Uma API REST trata-se de uma interface de programação de aplicações que, obedecendo a um conjunto de regras e restrições implementadas pela arquitetura REST, sigla para REpresentational State Transfer, que permite que a interação com serviços web seja mais efetiva.

Foi necessário implementar também a framework Jersey, que consiste numa framework open-source para o desenvolvimento de serviços web RESTful em linguagem Java. Foi também necessário recorrer ao Grizzly, que se define como uma framework NIO que torna possível a construção de servidores web. O Grizzly é também utilizado como um conector HTTP.

Para ser possível guardar os anúncios após o encerramento do servidor foi necessário recorrer a uma forma de armazenamento persistente, que neste caso é armazenamento numa base de dados PostgreSQL. De maneira a ser possível conectar o servidor à base de Dados recorreu-se ao JDBC, sigla para Java Database Connection, que é uma API que serve para conectar e executar queries com uma base de dados, existindo a necessidade de recorrer a drivers para o efeito.

De forma a tornar mais fácil transferir e usufruir de várias bibliotecas externas à linguagem Java, sem ter a necessidade de recorrer a drivers para o efeito (por exemplo ficheiros .jar), recorreu-se à plataforma Maven, que se trata de um software para construção e gerenciamento de projetos.

2 Implementação

Este trabalho está subdividido em dois diretótios, o diretório `client` e ainda o diretório `rest`. O diretório `client` contém dois ficheiros, `Client.java`, que é responsável por executar o cliente geral, e ainda o `ClientManager.java`, responsável por correr o cliente de gestão.

No caso do diretório `rest`, possui todos os ficheiros necessário para a correta execução do servidor, mais especificamente os ficheiros `Server.java` e `ServerResource.java`, sendo que o primeiro é responsável por abrir o servidor e o segundo por executar as operações relativas ao servidor (como por exemplo colocar e selecionar anúncios presentes na base de dados). Os ficheiros `Ad.java` e `Message.java` servem para organizar e facilitar os dados a serem enviados para o servidor que, por sua vez, é responsável por enviar para a base de dados. Por último existe ainda os ficheiros `ConnectionDB.java` e ainda o `Queries.java`. O primeiro serve essencialmente para abrir a conexão do servidor com a base de dados PostegreSQL, usufruindo de JDBC para o efeito, e o segundo possui todas as operações necessárias para interagir com a base de dados, isto é, executa as queries necessárias em ordem para obter aquilo que for necessário ou pedido.

3 Instruções de execução

1. Abra um terminal no diretório do servidor (`ServerSide`)
2. Execute o comando `mvn compile` no terminal. Se aparecer a mensagem "Build Success", então o ficheiro foi compilado com sucesso
3. Após compilar, execute o comando `mvn exec:java` de forma a inicializar o servidor na porta 8080.
4. Com o servidor em curso, abra outro terminal no diretório dos clientes (`ClientSide`)
5. Execute o comando `mvn compile` no terminal. Se aparecer a mensagem "Build Success", então o ficheiro foi compilado com sucesso
6. Após compilar o projeto, pode executar o comando `mvn exec:java -Dexec.mainClass=sd.client.Client` se pretender correr o cliente geral ou o comando `mvn exec:java -Dexec.mainClass=sd.client.ClientManager` se a escolha for o cliente de gestão.

7. Se pretender, pode saltar o ponto anterior e executar algumas das operações disponíveis através do browser:
   - As operações de registar um novo anúncio e enviar uma nova mensagem apenas podem ser executadas com sucesso através do terminal. Mas todas as restantes operações podem ser executadas tanto no terminal como no browser. Operações como listar anúncios ativos, procurar anúncios através de campos específicos para o efeito e verificar mensagens relativas a um determinado anúncio, operações relativas ao cliente; ou listar todos os anúncios e alterar o estado de um determinado anúncio, estas relativas ao cliente de gestão. Para executar as mesmas através do browser, é necessário aceder ao endereço `http://localhost:8080/server` e a partir daí podem ser executadas várias operações
     - Cliente geral - `http://localhost:8080/server`
       - listar anúncios ativos
         `http://localhost:8080/server/ads`
       - listar anúncios por descrição
         `http://localhost:8080/server/ads?description=<texto>`
       - listar anúncios por descrição e localização
         `http://localhost:8080/server/ads?description=<texto>&local=<xtexto>`
       - mostrar um determinado anúncio
         `http://localhost:8080/server/ads?aid=<aid do anúncio desejado>`
     - Cliente de gestão - `http://localhost:8080/server/manage`
       - listar todos os anúncios
         `http://localhost:8080/server/manage/ads`
       - alterar o estado de um anúncio
         `http://localhost:8080/server/manage/changead?state=<estado>&aid=<anúncio>`
8. Quando desejar, pode encerrar o servidor através de dois atalhos no terminal, Enter ou CTRL+C.

4 Discussão de resultados

A maior dificuldade sentida neste trabalho foi a experiência praticamente nula com REST e Spring, o que, no início, dificultou bastante e inclusivé atrasou quaisquer avanços e/ou progressos. Apesar disso, após uma pesquisa exaustiva, recorrendo sempre à documentação oficial, foi possível perceber melhor as funcionalidades e a forma como o próprio REST funciona, o que permitiu fazer avanços no trabalho.

A grande diferença na forma de comunicação entre o cliente-servidor do 1º para o 2º trabalho, foi uma das maiores surpresas deste trabalho, pois são duas maneiras de comunicar completamente distintas, o que obrigou a uma readaptação de forma a conseguir atingir o objetivo final, que era conseguir colocar as comunicações cliente-servidor a funcionar.

5 Conclusão

Este trabalho, quando comparado com o primeiro feito em Java RMI, foi um pouco mais desafiante, pois houve a necessidade de ter que se adaptar as comunicações cliente-servidor de forma a ser possível utilizar diferentes plataformas ou tecnologias, recorrendo para isso ao formato JSON, pois existem muitas tecnologias diferentes com a capacidade de interpretarem respostas em JSON.

Após utilizar duas tecnologias como o Java RMI e o REST, apesar de ambas servirem para o mesmo propósito, que é comunicação cliente-servidor, as diferenças entre ambas são enormes, o que se revela ser um ponto bastante interessante, pois seria difícil de imaginar que fosse possível existir tantas diferenças entre duas tecnologias cujo propósito final é semelhante.

Apesar das dificuldades sentidas, concluiu-se com sucesso esta tarefa pois atingimos o objetivo de criar um servidor funcional, com capacidade de comunicar com várias plataformas.

6 Webgrafia

`https://www.javatpoint.com/restful-web-services-spring-boot`
`https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api`
`https://pt.wikipedia.org/wiki/Spring_Framework`
`https://maven.apache.org/`
Saias, José in "Aulas Teóricas de Sistemas Distribuídos". 2022 at University of Évora
Saias, José in "Aulas Práticas de Sistemas Distribuídos". 2022 at University of Évora