1 Trabalho Prático de Sistemas Distribuídos
Por: Carlos Palma 46520

Introdução
Foi proposto pelo professor José Saias, responsável pela cadeira de sistemas distribuídos a elaboração de um trabalho que consistia na implementação das aplicações servidor, cliente geral e cliente de gestão que permitam, desde as mais variadas localizações, o acesso ao serviço.

Desenvolvimento
Após leitura do enunciado optei por utilizar Java RMI como solução de Middleware, pois podemos ter uma aplicação em que se invocam métodos de um objeto remoto, para que fosse possível o acesso ao serviço desde as mais variadas localizações. Para conexão com a BD e alterações/consultas da mesma foi utilizado JDBC- Java Database Connectivity que permite o acesso remoto através das suas interfaces.

Classes
- Cliente_geral – Responsável pela implementação da interface de cliente geral e criação do objeto remoto, são chamados os métodos necessários consoante as intenções do mesmo.
- Cliente_gestão – Responsável pela implementação da interface de cliente de gestão e criação do objeto remoto, são chamados os métodos necessários consoante as intenções do mesmo.
- Metodos – Interface Remota;
- MetodosImp – Objeto remoto utilizado para a invocação remota, implementa os métodos da interface remota e em cada um deles é realizada uma conexão à BD com algum intuito.
- PostgresConnector – Classe responsável pela conexão à BD Postgresql.
- Server – Classe que representa o servidor.

Métodos:
Irei de seguida resumir aquilo que acontece em cada método de cada classe:

Cliente_geral:
- main() - Obtém as configurações do ficheiro configs.properties e com as mesmas obtém o objeto remoto, de seguida apresenta o menu ao utilizador e consoante a intenção do mesmo, com o auxilio de um switch chama o método do objeto remoto necessário para realizar a operação pretendida.

Cliente_gestao:
- main() - Muito semelhante ao que acontece na classe Cliente_geral, no entanto são utilizados outros métodos pois as intenções do cliente de gestão são diferentes das do cliente geral.

MetodosImp:
- registo(String tipo,String localizacao,int preco,String genero,String anunciante,String tipologia) – Realiza a conexão à base de dados, consulta a BD e verifica qual o maior aid da tabela e adiciona 1, tendo em conta que os aids são primary keys identificativas de cada registo e não podem existir iguais, de seguida realiza um insert com os dados recebidos como parâmetros(enviados pelo cliente), todos os anúncios são considerados inativos aquando do registo,por fim desconecta-se da BD e retorna uma string a indicar que o anuncio foi registado.
- listar_oferta() - Realiza a conexão à base de dados, realiza um select para consultar quais os registos na tabela de oferta e adiciona os mesmos a uma ArrayList, por fim desconecta-se da BD e a ArrayList é retornada.

- listar_procuro() - Semelhante ao método listar_oferta() mas ao invés de verificados os registos de oferta, são consultados os registos de procura.
- listar_anunciante(String anunciante) - Realiza a conexão à base de dados, realiza um select para consultar quais os registos da tabela que têm como anunciante a string recebida como parâmetro, adiciona os registos a uma ArrayList sempre que tal se verifique, por fim desconecta-se da BD e retorna a ArrayList.
- list_aid(int aid) - Realiza a conexão à base de dados, realiza um select para consultar qual o registo da tabela que têm como aid o inteiro recebido como parâmetro, cria uma string e insere os detalhes do registo na mesma, por fim desconecta-se da BD e retorna a string.
- enviar_msg(int aid, String msg) – Realiza a conexão à base de dados e realiza um um insert do aid recebido como parâmetro e da mensagem também recebida por parâmetro. Por fim desconecta-se da BD e retorna uma string indicando que a mensagem foi enviada.
- consultar_msg(int aid) – Realiza a conexão à base de dados, realiza um select para consultar quais as mensagens que têm como aid o inteiro recebido como parâmetro, adicionando as mensagens a uma ArrayList sempre que tal se verifique, por fim desconecta-se da BD e retorna a ArrayList com as mensagens.
- listar_inativos() - Realiza a conexão à base de dados, realiza um select para consultar quais as registos que têm como estado inativo, adicionando os mesmos a uma ArrayList sempre que tal se verifique, por fim desconecta-se da BD e retorna a ArrayList com os registos.
- ativar_anuncio(int aid) – Realiza a conexão à base de dados, realiza um update set de forma a alterar o valor na coluna estado para ativo onde o aid é igual ao recebido como como parâmetro, por fim desconecta-se da BD e retorna uma string a indicar que o anuncio foi ativado.
- alterar_estado(int aid, String estado) – Realiza a conexão à base de dados, realiza um update set de forma a alterar o valor na coluna estado para o valor recebido como parâmetro, onde o aid é igual ao recebido como parâmetro, por fim desconecta-se da BD e retorna uma string a indicar que o estado do anuncio foi alterado.

Tabelas da Base de Dados

- t1 – Tabela criada de forma a armazenar os dados do anuncio, tem como colunas:
  - aid (integer) que é primary key e corresponde aos aids identificativos dos registos
  - localizacao (varchar)
  - preco (integer)
  - data (timestamp without time zone)
  - anunciante (varchar)
  - tipologia (varchar)
  -tipo (varchar)
  -estado (varchar)
- m – Tabela criada de forma a armazenar as mensagens enviadas para cada aid, tem como colunas:
  - aid (integer)
  - mensagens (varchar)

Conclusão

Considero que foi um trabalho desafiante pelo simples facto da única aula prática a que não assisti ter sido a de Java RMI, tive algumas dificuldades iniciais a tratar do servidor que foram rapidamente ultrapassadas com uma leitura daquilo que aconteceu durante a respetiva aula. De resto é sempre um prazer trabalhar com Java, uma linguagem de programação super intuitiva e com inúmeras funcionalidades, o que facilita sempre o meu trabalho enquanto aluno. Aprendi bastante com este trabalho e acredito ter implementado tudo aquilo que foi pedido pelo professor da melhor forma que consegui.