Relatório 2 Trabalho de Sistemas Distribuídos

Por: Carlos Palma 46520

Introdução

Foi pedido pelo professor José Saias a elaboração de um sistema para gerir os anúncios de quartos para arrendamento mas desta vez com arquitetura distribuída e com as seguintes características:

1. Cliente geral
a) Registar novo anúncios (oferta ou procura), sendo que o anúncio fica em estado inativo até ser aprovado pelo gestor. O sistema atribui um novo código único, que será devolvido como resultado da operação. Cada anúncio terá localização, preço, género, data, anunciante, tipologia (quarto, T0, T1…) e poderá ser de oferta de alojamento, ou de procura. Tem ainda um estado (inativo, ativo), preço e descrição.
b) Listar anúncios (com estado ativo) de ambos os tipos.
c) Procurar anúncios, enviando texto a pesquisar na descrição, e opcionalmente uma localização.
d) Obter todos os detalhes de um anúncio, dado o seu identificador (aid).
e) Enviar nova mensagem ao anunciante, pelo identificador do anúncio (aid). E consultar as mensagens inseridas para um determinado anúncio (aid).

2. Cliente de Gestão
a) Listar anúncios por estado.
b) Aprovar um anúncio, ou alterar o estado de um anúncio (ativo/inativo).

3. Serviço em backend (servidor)
a) Serviço para as operações a disponibilizar a cada cliente.
(nas listagens do cliente geral, mostrar apenas anúncio em estado ativo)
b) Armazenamento persistente de dados

Outros aspectos:
Abstração dos detalhes de comunicação, usando formas de comunicação mencionadas nas aulas e não dependentes de uma tecnologia específica.
Armazenamento persistente com uma BD em Postgres.
A interação pode fazer-se com uma aplicação de linha de comandos.
Quaisquer parâmetros de configuração devem estar fora do código, sendo passados como argumento à aplicação ou lidos de um ficheiro de propriedades (ver java.util.Properties).
A solução implementada deve ser compatível com a plataforma de alunos.di.uevora.pt.

Desenvolvimento

De formo a ir ao encontro daquilo que foi pedido resolvi criar um projeto em Java, utilizando REST com o framework Jersey, este trabalho foi maioritariamente baseado na atividade 11.1 desenvolvida na aula em que é usado Jersey que é um framework Java que implementa Java API for RESTful Web Services (JAX-RS) para a construção de serviços RESTful.

Classes

- Cliente_geral - Classe responsável pela implementação da interface de cliente geral e realização dos pedidos.
- Cliente_gestão - Classe responsável pela implementação da interface de cliente de gestão e realização dos pedidos.
- Anuncio – Classe responsável pela representação de um anuncio.
- AnuncioResource – Classe responsável pelo tratamento e atendimento dos pedidos das classes Cliente_geral e Cliente_gestão.
- DbOperations – Classe responsável pelas interações com a base de dados, realiza consultas e inserts.
- MainAppServer – Classe responsável pela representação do servidor.
- Mensagem – Classe responsável pela representação de uma mensagem.
- PostgresConnector – Classe responsável pela ligação à base de dados.

Métodos

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

Cliente_geral:

- menu - Mostra ao utilizador o menu de opções e lê a opção escolhida pelo utilizador.
- postAnuncio – Abre uma conexão HTTP e realiza um pedido POST ao servidor de modo a registar um novo anuncio com base nos argumentos recebidos recorrendo a json.
- getAnuncios - Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a obter os anuncios com estado ativo recorrendo a json.
- getAnunciosTipo – Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a obter os anuncios ativos de um certo tipo recorrendo a json.
- procurarAnuncios – Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a obter os anuncios ativos com base nos argumentos recebidos, descrição e localização recorrendo a json.
- procurarAnuncioAid – Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a obter os anuncios ativos com base no argumento recebido, aid recorrendo a json.
- postMensagem - Abre uma conexão HTTP e realiza um pedido POST ao servidor de forma a registar uma nova mensagem com base nos argumentos recebidos recorrendo a json.
- consultarMensagemsPorAid - Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a obter as mensagens com base no aid recebido como argumento recorrendo a json.
- main – Responsável por realizar as interações com o utilizador, chama o método menu e consoante a opção escolhida pelo utilizador recorre a um switch para escolher o método a utilizar e ler os inputs do utilizador.

Cliente_gestao

- menu - Mostra ao utilizador o menu de opções e lê a opção escolhida pelo utilizador.
- getAnunciosAtivos- Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a obter os anuncios com estado ativo recorrendo a json.
- getAnunciosInativos - Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a obter os anuncios com estado inativo recorrendo a json.
- ativarAnuncio - Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a alterar o estado de um anuncio para ativo com base no argumento aid recebido, recorrendo a json.

- desativarAnuncio - Abre uma conexão HTTP e realiza um pedido GET ao servidor de forma a alterar o estado de um anuncio para inativo com base no argumento aid recebido, recorrendo a json.
- main - Responsável por realizar as interações com o utilizador, chama o método menu e consoante a opção escolhida pelo utilizador recorre a um switch para escolher o método a utilizar e ler os inputs do utilizador.

Anuncio

- Não vou detalhar completamente todos os métodos desta classe pois consiste na maioria em variáveis, construtores, gets e setters com aquilo que é necessário para representar um anuncio.

Mensagem

- Não vou detalhar completamente todos os métodos desta classe pois consiste na maioria em variáveis, construtores, gets e setters com aquilo que é necessário para representar uma mensagem.

AnuncioResource

- criarAnuncio – Retorna um bool true ou false dependendo do sucesso da operação de inserção do anuncio na base de dados recorrendo ao método registarAnuncio de dbOperations enviando o anuncio recebido como argumento.
- listaAnunciosAtivos – Retorna uma lista de anúncios ativos recorrendo ao método listarAnuncios de dbOperations enviando a string "ativo" a indicar o estado pretendido.
- listaAnunciosInativos – Retorna uma lista de anúncios inativos recorrendo ao método listarAnuncios de dbOperations enviando a string "inativo" a indicar o estado pretendido.
- listarAnunciosTipo - Retorna uma lista de anúncios recorrendo ao metodo listarAnunciosPorTipo de dbOperations enviando a string tipo recebida como argumento.
- porcurarAnuncios - Retorna uma lista de anúncios recorrendo ao metodo procurarAnuncio de dbOperations enviando as strings descricao e localizacao recebidas como argumento.
- ativarAnuncio - Retorna um bool true ou false dependendo do sucesso da operação de alteração do estado de um anuncio na base de dados recorrendo ao método ativarAnuncio de dbOperations enviando a string aid recebida como argumento.
- desativarAnuncio - Retorna um bool true ou false dependendo do sucesso da operação de alteração do estado de um anuncio na base de dados recorrendo ao método desativarAnuncio de dbOperations enviando a string aid recebida como argumento.
- enviarMensagem - Retorna um bool true ou false dependendo do sucesso da operação de inserção de uma mensagem na base de dados recorrendo ao método enviarMensagem de dbOperations enviando mensagem recebida como argumento.
- consultarMensagens – Retorna uma lista de mensagens recorrendo ao método consultarMensagens de dbOperations enviando a string aid que foi recebida como argumento.

DbOperations

- setDbProperties – Recorre ao ficheiro de configurações da bd de forma a obter as informações necessárias para realizar a ligação à base de dados.
- registarAnuncio - Recebe um anuncio como argumento e utiliza o mesmo de forma a realizar um insert na base de dados com base no mesmo.
- listarAnunciosPorTipo – Recebe uma string tipo como argumento e realiza um select na tabela anuncios da bd com o estado ativo e o tipo indicado retornando uma lista de todos os anuncios com esse estado e tipo.

- listarAnuncios – Recebe uma string estado como argumento e realiza um select na tabela anuncios da bd com o estado indicado retornando uma lista de todos os anuncios com esse estado.
- procurarAnuncio - Recebe duas strings descricao e localizacao como argumento e realiza um select na tabela anuncios da bd com o estado ativo e o tipo indicado retornando uma lista de todos os anuncios que contem parte da string descricao na coluna descricao da tabela ou localizacao igual à localizacao recebida.
- procurarPorAid – Recebe um inteiro aid como argumento e com base no mesmo realiza um select retornando uma lista com o anuncio com aquele aid.
- ativarAnuncio – Recebe uma string aid e como argumento e com base no mesmo realiza um update colocando o estado como ativo na linha que tem o aid recebido.
- desativarAnuncio - Recebe uma string aid e como argumento e com base no mesmo realiza um update colocando o estado como inativo na linha que tem o aid recebido.
- enviarMensagem - Recebe uma mensagem como argumento e utiliza o mesmo de forma a realizar um insert na base de dados com base no mesmo.
- consultarMensagens – Recebe uma string aid como argumento e com base na mesma realiza um select na tabela mensagens retornando uma lista das mensagens que têm aquele aid.

MainAppServer

- getBaseURI – Devolve o URI onde vai ficar o serviço.
- startServer – Ativa um serviço com os REST resources existentes neste pacote.

Tabelas e base de dados

Foram criadas as seguintes tabelas para representação dos anuncios e mensagens na base de dados, adicionei as mesmas a uma pasta no projecto para facilitar a criação das mesmas caso seja necessário.

```
CREATE TABLE anuncios (
   aid serial PRIMARY KEY,
   estado varchar(10) NOT NULL,
   tipo varchar(10) NOT NULL,
   tipologia varchar(10) NOT NULL,
   genero varchar(15) NOT NULL,
   descricao varchar(500),
   preco varchar(100),
   anunciante varchar(10) NOT NULL,
   contacto varchar(50),
   localizacao varchar(50),
   data varchar(50) NOT NULL
);

CREATE TABLE mensagens(
   aid integer,
   remetente varchar(20),
   mensagem varchar(500),
   FOREIGN KEY (aid) REFERENCES anuncios(aid)
);
```

Extras

Foi implementada alguma redundância no backend/servidor, para reforçar a disponibilidade do serviço, tolerando falhas, para tal o serviço é iniciado em vários URI's.

Conclusão

Mais uma vez tratou-se de um trabalho super desafiante e divertido de elaborar, sempre que existe uma questão também existe uma resposta nos recursos fornecidos pelo professor, seja aulas teóricas ou práticas o que sem duvida facilita bastante o nosso trabalho. Fiquei triste por não conseguir implementar todos os extras propostos mas em suma acredito que se trata de um bom trabalho pois faz tudo aquilo que é pedido com base.