Unidade Curricular Integração de Sistemas

2019 - 2020

Trabalho Prático Nº2

Three-tier Programming with Object-Relational Mapping

Alexandre Américo Monteiro Ferreira | **nº 2016240875** | **LEI**

Tomás Faria Martins | **nº 2016234128** | **LEI**

Introdução

Neste projeto usamos a arquitetura Three-Tier Architecture que consiste numa arquitetura cliente-servidor modular constituída por três camadas, entre elas a camada de apresentação, a camada da aplicação também denominada como camada lógica e a camada de dados. Estas três camadas permitem que a aplicação trabalhe sobre três sistemas independentes, sendo que na camada de apresentação estão os clientes em execução, na camada lógica executam-se os processos lógicos em servidores remotos ou locais e na camada de dados trata-se tudo o que está relacionado com o gerenciamento de dados da aplicação.

A comunicação entre as camadas é feita de forma segura e ordenada. A camada de apresentação nunca comunica diretamente com a camada de dados, para fazer esta ligação existe a camada lógica que processa o pedido feito pela camada de apresentação, comunica com a camada de dados para executar o pedido e depois transmite uma resposta à camada de apresentação.

Esta arquitetura permite que um cliente seja executado em qualquer sistema operacional e que este comunique apenas com a camada lógica. A camada de dados pode ser de diferentes tipos e pode possuir diversos designs desde que a camada lógica saiba como comunicar e trabalhar com dos dados.

O nosso projeto consiste numa aplicação web para vender artigos em segunda mão chamada MyBay, e para isso usamos diversas tecnologias, entre elas o WildFly Application Server para fazer o *deploy* do nosso projeto; uma ferramenta de ORM (Object/Relational Mapping) denominada Hibernate com Java Persistance API para realizar a relação dos objetos com os dados que os mesmo representam; a base de dados usada foi o MySQL e o IDE usado foi o IntelliJ com recurso ao Maven.

Camada de Apresentação

Esta camada tem a responsabilidade de apresentar a aplicação ao cliente e de realizar todas as interações com o mesmo permitindo ao cliente interagir com as funcionalidades da segunda camada de forma segura e intuitiva. Um cliente nunca contacta diretamente com a terceira camada, todas as ações do cliente são emitidas e tratadas na camada lógica que por sua

vez verifica a camada de dados e executa as tarefas necessárias, enviado a resposta de novo a camada de apresentação.

No nosso projeto implementamos um Servlet que é ativado quando o utilizador executa uma ação, e na sua inicialização é criada uma HTTP Session que guarda algumas ações que são executadas pelo utilizador (por exemplo, para verificar que o utilizador foi autenticado ou não, quando este se autêntica guardamos na session que o utilizador está autenticado. Para realizar a autenticação o utilizador tem de ir à página de Login e inserir o seu e-mail e a sua password (esta password é encriptada com sha-236 (Secure Hash Algorithm) e guardada na base de dados). O Servlet é ativado com métodos "doPost" e "doGet" que são emitidos pela interface consoante as ações do utilizador. Após isso a servlet redireciona o utilizador para determinadas páginas (por exemplo, quando o utilizador quer realizar uma procura) ou se precisar, esta recorre a métodos definidos na interface local para satisfazer os pedidos do utilizador (por exemplo, quando o utilizador pretende realizar login a servlet tem de contactar a camada lógica para esta confirmar se o utilizador está ou não registado na aplicação).

Em termos de interface, usamos ficheiros jsp para criar toda a interface da aplicação e para facilitar a criação de código HTML, recorremos a uma ferramenta online chamada BootStrap. Na figura 1 podemos analisar o fluxo que a nossa aplicação. É de salientar que no menu dos "MeusItems" o utilizador pode editar e remover os items que possui assim como no menu "EditarPerfil" o utilizador pode editar o seu perfil e apagar a sua conta.

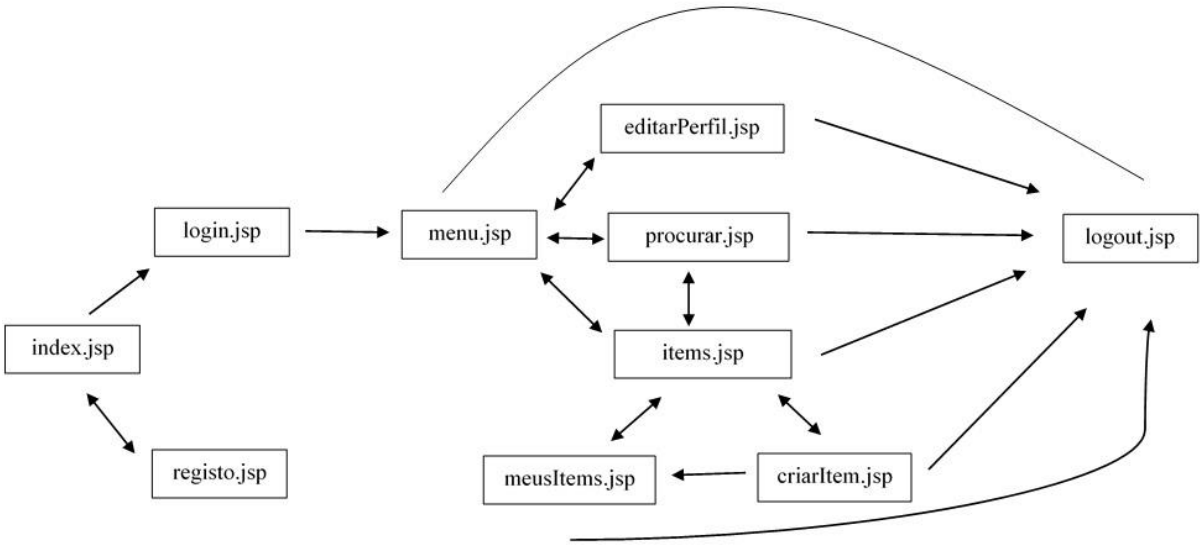

Figura 1 - Diagrama de fluxo da aplicação

A interface vai de encontra com a figura 2, em que todas as interfaces possuem uma barra de navegação (que se encontra no topo da página), e consoante a interface podemos ter caixas de inputs para inserir dados (por exemplo, alterar as informações do utilizador ou criar item (figura 2)) com botões para salvar as alterações ou então uma lista de items (que resultam da procura que o utilizador executa, ou quando o utilizador vai à aba "Meus Items").

A parte mais complexa de implementar foi o menu das procuras em que neste menu era necessário listar os items e fornecer ao utilizador forma de ordenar a lista de items que possuía. Posto isto, para fazer a listagem dos items utilizamos um componente do BootStrap chamado "Card" que permite inserir a foto e as informações relativas ao item, para realizar a ordenação fizemos uso do request para guardar a pesquisa que o utilizador tinha realizado para depois então poder ser feita uma ordenação ao gosto do utilizador (por nome, preço, data de publicação na forma ascendente e descendente).

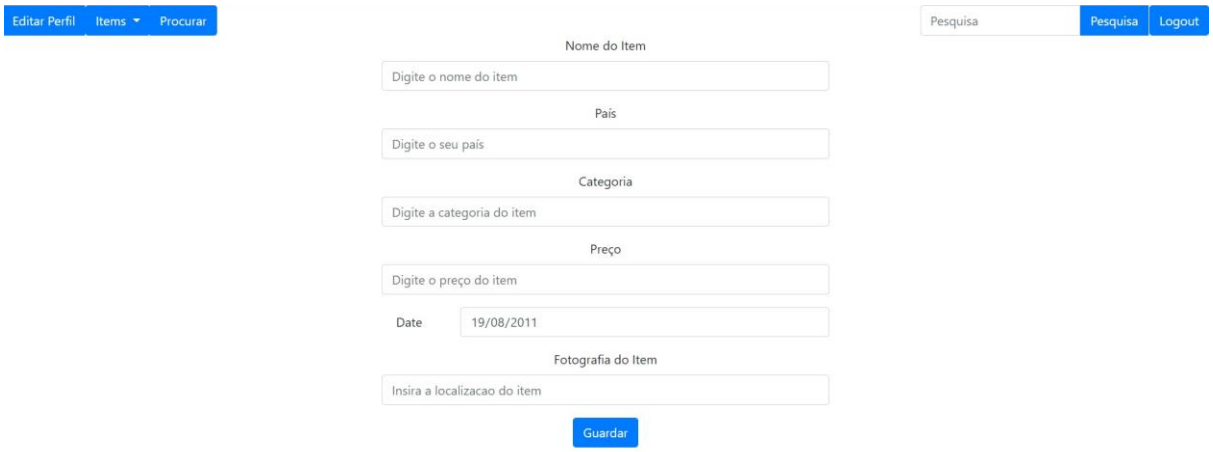

Figura 2 - Interface para a criação de um item

Camada de Dados

Nesta camada são definidos os objetos que vamos utilizar no programa. Através dos comandos da figura 3 são geradas tabelas na base de dados com os respetivos atributos de cada classe criada. Tendo em conta as especificações do problema criamos duas classes: Item e User. Ambas as tabelas têm uma chave primária (um id) que é definida através dos comandos

presentes na figura 4. Para além do id, a classe User possui nome, país, email e password e a classe Items é constituída por um nome, país, categoria, data de publicação e foto. Para a construção dos objetos, no caso do User é necessário preencher todos os campos enquanto que no caso do Item é opcional inserir ou não o url da foto (figura 5 e 6). Como um utilizador pode possuir vários itens, a relação entre a classe User e a classe Item é de One-to-Many. Desta forma, a tabela "Items" da base de dados irá conter a chave primária da tabela "Users" como chave estrangeira. De forma a criar esta relação, o normal seria adicionar um campo User no Item e um campo List<Item> no User e colocar em cima destes atributos @One-to-One e @One-to-Many, respetivamente. Porém com esta solução obtivemos o seguinte problema: na base de dados não estava a assumir o id do utilizador como chave estrangeira na tabela Item, logo, não conseguimos estabelecer ligação entre as tabelas. Para contornar isso, criámos um campo user_id que é atribuído automaticamente quando o utilizador cria um objecto visto que o seu id está presente na sessão.

```
@Entity
@Tabel(name="Users")
public class Item implements Serializable{
```

Figura 3 - Definem a classe como entidade e atribuem o nome da respetiva tabela na base de dados.

```
@Id
@GeneratedValue (strategy=Generatetype.Auto)
```

Figura 4 - Servem para definir uma variável como chave primária da tabela e gerar um valor automático para esse atributo.

```java
@Id
@GeneratedValue (strategy=GenerationType.AUTO)
   private int id;
   private String name;
   private String country;
   private String mail;
   private String password;


   public User () {
      super ();
   }

   public User (String name, String country, String
mail, String password) {
      super ();
      this.name = name;
      this. country = country;
      this.mail = mail;
      this.password = password;
   }
```

Figura 5 - Atributos e construtores da classe User.

```java
   @Id
   @GeneratedValue (strategy=GenerationType.AUTO)
   private int id;
   private String name;
   private float price;
   private String country;
   private String category;
   private String publish_date;
   private String foto;
   private int user_id;


   public Item () {
      super ();
   }

   public Item (String name, float price, String country,
String category, String foto, int user_id, String data) {
      super ();
      this.name = name;
      this.price = price;
      this.country = country;
      this.category = category;
      this.foto = foto;
      this.user_id = user_id;
      this.publish_date = data;
   }

   public Item(String name, float price, String country,
String category, int user_id, String data){
      super();
      this.name = name;
      this.price = price;
      this.country = country;
      this.category = category;
      this.user_id = user_id;
      this.publish_date = data;
   }
```

Figura 6 - Atributos e construtores da classe Item.

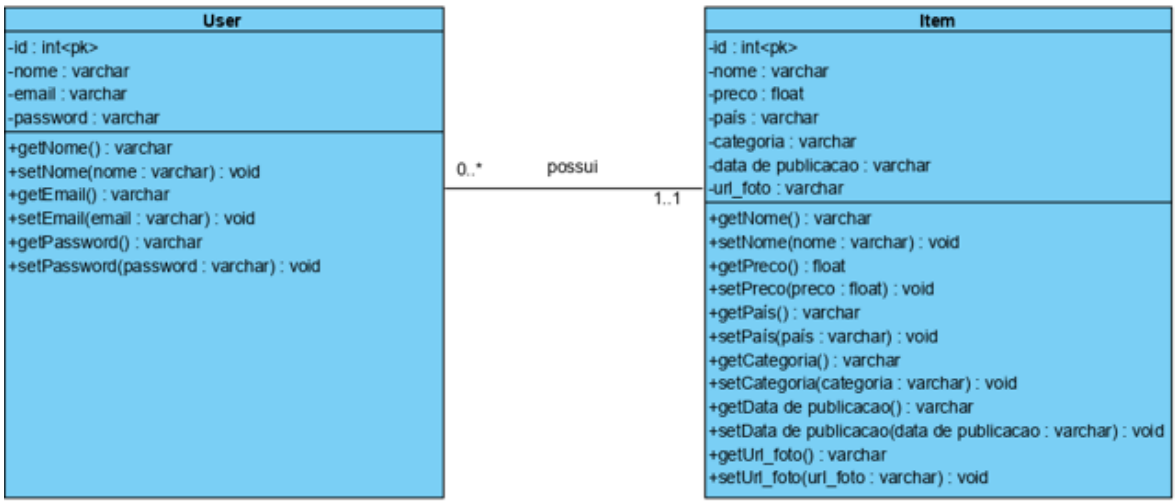

Figura 7 - Diagrama ER

Camada Lógica

Corresponde à camada definida em Enterprise JavaBeans (EJB) que tem como finalidade fazer o tratamento dos dados do programa. Sendo assim, esta é a única camada que tem acesso aos serviços da camada de dados. Isto é feito através de uma variável EntityManager que permite adicionar ou remover elementos à base de dados e procurar na mesma através da execução de queries (figura 8, 9 e 10). Nela estão contidas todas as funções que permitem ao programa responder aos pedidos do utilizador. As funcionalidades a que o cliente tem acesso são definidas numa interface local (figura 11) para posteriormente poderem ser chamadas a partir da camada de apresentação (com o uso da Servlet). Como já foi referido anteriormente, quando o utilizador realiza um pedido ao programa através da página web, é acionada uma action que vai ser reconhecida na Servlet. Mediante essa action é chamada na Servlet a função da camada lógica que permite realizar a operação pretendida. O único tipo de informação que os utilizadores têm acesso é a informação parcial (não é mostrado o id do item nem do dono) dos items. O utilizador tem acesso às informações do items através de pesquisas, que serão expostas na página web. Visto que vários utilizadores podem aceder a esta camada em simultâneo seria necessário implementar mecanismos que gerissem as transações de informação porém os EJB são capazes de fazer isso autonomamente. Por esta mesma razão optamos por definir o EJB como stateless caso contrário seria necessário adicionar novos beans visto que uma comunicação stateful bloqueia o JavaBean quando este está a ser acedido por um utilizador.

Entre a camada lógica e a camada de apresentação nunca circulam objetos do tipo User ou Item. Sempre que é necessário aceder a um Item/User a camada de apresentação envia à camada lógica o id (do item, user ou ambos) e a camada lógica carrega toda a informação que necessita. Quando é necessária apresentação de informação na camada de apresentação, a camada lógica cria uma string (com a informação necessária) e a na camada de apresentação a string é “desmontada” e a informação é apresentada (por exemplo quando é necessário apresentar os items que um determinado utilizador possui).

Entre a camada lógica e a camada de dados, quando é necessário informação a camada lógica constrói uma querie de modo a obter única e exclusivamente a informação que quer e guarda a informação numa lista de Users ou de Items consoante o que pretende.

```
private EntityManager em;
```

Figura 8 - Definição da variável Entity Manager

```
@Local
public interface UserEJBLocal {

    int registaUser(String email, String password, String nome, String pais);

    int loginUser(String email, String password);

    void editUser(int id, String nome, String pais, String password);

    public List<Item> pesquisa(String pesquisa, String pesquisa2, String nome,
String flag);
    public List<Item> getUser_items(int id_user);

    void removeUser(int id);

    void addItem(String nome, String preco, String pais, String categoria, String
url_foto, int user_id, String data);

    void editItem(int id_item, int id_user, String nome, String preco, String pais,
String categoria, String url_foto);

    public List<Item> pesquisa_rapida(String pesquisa);

    int removeItem(int id_item, int id_user);

    List <Item> pesquisa_ordenada(String pesquisa, String pesquisa2, String
tipo_pesquisa, String tipo_ordem, String parametro_pesquisa, String nome,
boolean flag);

}
```

Figura 11 - Métodos presentes na Interface local da camada lógica.

"Gestão" do projeto

O projeto encontra-se dividido em quatro pastas: a pasta "business" contém a parte lógica do programa, nomeadamente a interface local e os JavaBeans; a pasta "data" contém as classes que são criadas para construir a base de dados e um ficheiro "persistance.xml" onde definimos o valor da "EntityManager" usado na camada lógica e estabelecemos as propriedades do hibernate e do persistance para poder fazer a conexão com a base de dados; a pasta "web" contém a Servlet que permite a conexão entre o cliente e a camada lógica, e os ficheiros ".jsp" que constroem toda a interface; finalmente temos a pasta "ear" que é onde fica gravado o ficheiro.ear que vai usado pelo WildFly para dar deploy de todo o projeto. Nos ficheiros ".pom" instalamos todas as dependências e plugins necessários para a realização do projeto com as tecnologias usadas.