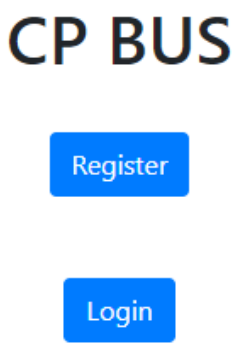
Integração de Sistemas 2021/22

Docente: Filipe João Boavida Mendonça Machado de Araújo

Assignment #2 - Three-tier Programming with Object-Relational Mapping



Miguel Lopes, 2021182710

Miguel Pimenta, 2018287956

Introdução

No âmbito da Unidade Curricular de Integração de Sistemas, foi nos pedido, que desenvolvêssemos uma "aplicação" que implementasse três camadas e que interajam entre si, temos então:

Presentation Tier - Onde foi desenvolvida a parte de front end da aplicação usando JSP;

Business Tier - Onde desenvolvemos as diferentes funcionalidades da aplicação através de EJB e beans;

Data Layer - Parte do programa onde são guardados os dados relativos às diferentes entidades. Estes dados são guardados numa base de dados através do java "persistence".

A aplicação em si, consiste em simular uma empresa de autocarros, onde permite aos clientes criar uma conta e comprar bilhetes, carregar dinheiro na sua carteira, ver as suas viagens, etc. Por outro lado, também temos um administrador, onde a sua função é gerir esta empresa, nomeadamente: ver quem são melhores clientes(top 5), criar novas viagens, apagar viagens existentes, ver os passageiros de uma determinada viagem, etc..

Presentation Tier

Na camada de apresentação recorremos a JavaServer Pages (JSP) para desenvolver a interface gráfica do projeto.

Quanto à estrutura das webpages em si, recorremos a um filter (SecurityFilter), onde a sua função é restringir o acesso do user à aplicação em si, no caso de este não ter feito o seu registo por login apriori. Para isso criámos uma pasta(secured), onde em comunicação com o filter, terá todas as funcionalidades bloqueadas até o user fazer login, ou registar-se na aplicação.

Na criação da interface gráfica do projeto usamos várias bibliotecas, como o caso de "bootstrap" para manter o design das JSP mais apelativa ao utilizador. Nesta camada de apresentação temos várias funcionalidades, que é o caso por exemplo de carregar a Wallet, esta permite ao utilizador, carregar a sua carteira com determinado valor pretendido. Todas estas funcionalidades, são representadas através de ficheiros JSP. De modo a que estas funcionalidades possam ser executadas, é necessária a atuação por parte de Servlets. Servlets são como pequenos servidores cujo objetivo é receber uma requisição HTTP, processá-la e responder ao cliente no nosso caso em um ficheiro JSP.

De modo a haver troca de informações entre Servlets, é usado o session management como forma de aceder a informações que vão ser comuns em diferentes partes do programa. É através deste acesso à "session" que são feitas algumas validações dos valores de input.

Os maiores problemas enfrentados foram a implementação das mensagens de erro presentes (e.g. erro ao fazer login), pois é necessário realizar na mesma página um "request" a um Servlet e definir esse atributo através de um ciclo if/else no próprio jsp. Outro problema foi a passagem de valores do Servlet para os ficheiros JSP. Infelizmente, não conseguimos fazer com que a palavra-passe fosse armazenada através de hash, estando apenas em "plain text".

Em seguida é ilustrado o Menu Principal (página após o login) que mostra a nossa interface gráfica (por parte do cliente):

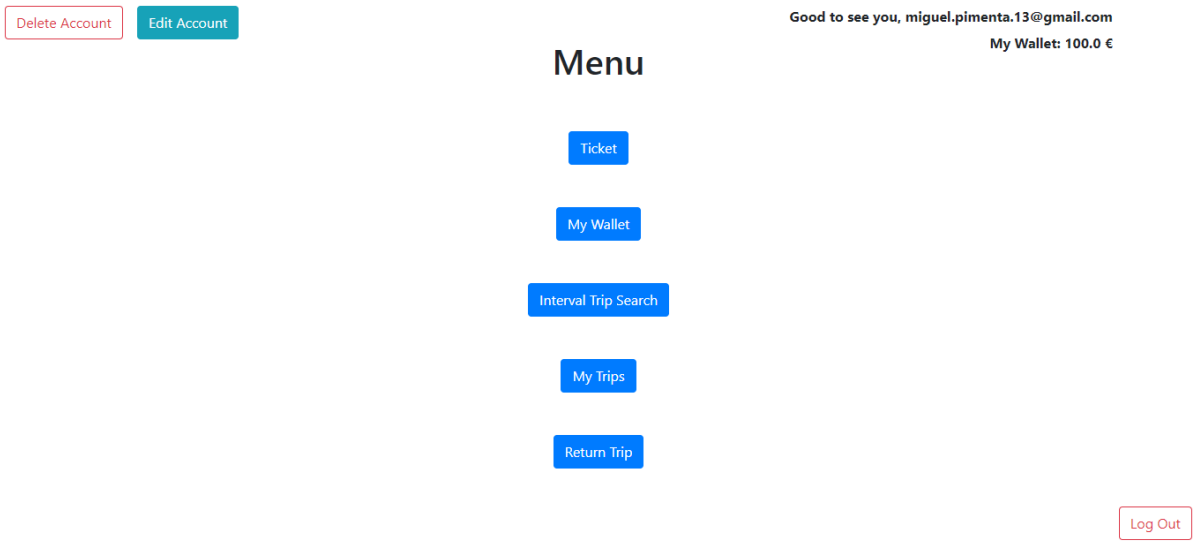


Figura 1: Interface do Menu Principal

Business Tier

Quanto ao “Business Logic Layer”, é neste nível que o nosso projeto contém os diferentes componentes EJBs e as diferentes funções/regras de negócio.

Em ambos os “beans” (manageClients, manageTrip) são realizados logs para verificar a atividade de cada funcionalidade, bem como, garantir a execução da mesma. Estes ficheiros “beans” têm como função, fazer a ligação entre a Presentation Tier e a Data Tier, pois é através deles, que através de parâmetros de entrada vindos do Servlet, é possível retornar o resultado pretendido para a JSP pretendida.

1. ManageStudents e IManageStudents

“ManageStudents” é referente a um “bean”, sem estado (“stateless”). É nestes “Session Beans” que são realizadas as funções chamadas ao longo da aplicação. No caso do nosso projeto, este vai ser responsável pela gestão de clientes. Para explicar melhor o funcionamento deste “beans” vamos analisar a função ChargeWallet(). Esta função tem como objetivo adicionar dinheiro à carteira do cliente. Para fazer isso, iremos ter de executar um update na base de dados relativamente aos dados deste utilizador. Para isso, o Servlet irá mandar o id do utilizador que está em sessão, bem como a quantidade de dinheiro a depositar na wallet para a função ChargeWallet() através da chamada do objeto beans manageClient. A função irá então através da conexão à BD (através de JDBC), atualizar os dados desse cliente. É necessário então, haver esta conexão entre Business Layer e Data Layer. Após esta atualização nos dados do cliente, estes são transportados para a camada de apresentação através de Servlets.

2. ManageTrips e IManageTrips

Quanto ao componente EJB “ManageTrips”, este é referente a um “bean”, sem estado (“stateless”), onde são realizados diferentes métodos. Estes métodos, são referentes à gestão de viagens e a todos os métodos que são influenciados pela entidade Viagem, tais como: comprar bilhetes, remover viagens, procurar viagens, etc.

Referente ao transporte dos dados estes são feitos de forma semelhante ao já explicados em cima, no tópico referente ao “ManageStudents”.

Data Tier

Em relação à camada de dados, estes são armazenados em Base de Dados Postgresql. Este armazenamento é realizado através de JPA - Java Persistence API (onde também é utilizado hibernate, que é uma framework para "object/relational mapping").

Foram criadas várias classes em Java de modo a estruturar o programa: Cliente, Viagem, Bilhete, classes estas que serão transformadas em entidades na Base de dados onde irão ser guardados os dados referentes às mesmas, como representado na figura 2.

A atualização dos dados guardados na Base de dados, é feita através de ligações da camada de Business Logic (EJB) onde através de JDBC são realizadas queries, onde iram enviar instruções de atualização à Base de Dados. Em seguida apresentamos o nosso Entity-Relational Diagram:

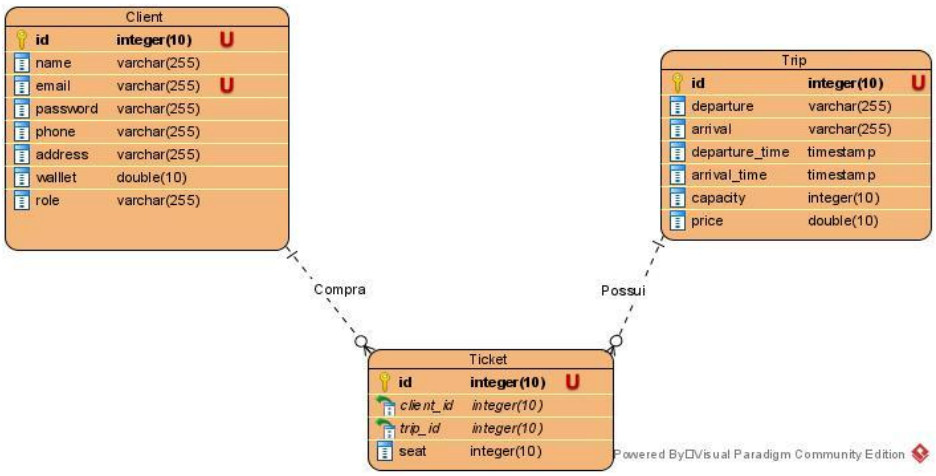


Figura 2: Entity Relationship Diagram

Como é possível observar, temos 3 classes, a classe Client, possui a chave primária "id", o email (único para cada registo) e os restantes atributos são os detalhes de cada conta do utilizador. Através desta classe é enviada uma "foreign key" para a Classe Ticket, com um relacionamento One-to-Many/Many-to-One, ou seja, um cliente pode ter vários tickets, mas um ticket só pode ter um cliente associado. É possível verificar na Classe "Client.Java" o atributo "mappedby" pois significa que o Client é a entidade que "manda" no relacionamento.

Associado à entidade "Ticket", esta possui um "id" (chave primária), "client_id" (foreign_key do cliente) e trip_id (foreign_key da Trip) - com uma relação Many-to-One/One-to-Many, ou seja, existem Tickets associados à mesma viagem, mas um Ticket só pode ter uma viagem associada, onde a classe responsável é a "Trip". Por último, ainda possui um "seat", que representa o lugar que o cliente irá ocupar durante a viagem.

Em relação à classe Trip, esta possui um id (chave primária), que é enviado para o Ticket (já explicado), um arrival/departure local, um arrival/departure time, a capacidade de pessoas que a viagem pode ter, e o preço associado à viagem.

Project Management and Packaging

Quanto ao projeto em si e à sua estrutura, temos as várias camadas (módulos) já referidas anteriormente divididas pelas diferentes páginas (web, ejbs, ear, devcontainer (containers para o Docker), jpa, rest (não utilizado com forte relevância neste projeto). Para criar todos estes módulos e obter todas as dependências necessárias para correr o nosso programa, usámos o livro do docente da cadeira para completar o ficheiro "pom.xml", ficheiro este que é fundamental no Maven, devido ao facto, de conter as informações e configurações/propriedades do projeto para o Maven criar o projeto (build).

Neste ficheiro está presente o packaging do "pom", o nome do projeto e o modelVersion (4), "maven.ear" (para gerar o ficheiro Java EE Enterprise Archive), "maven.jar" (JavaArchives - utilizado para distribuir as diferentes classes Java) e até "maven.ejb" (para gerar o ficheiro EJB), e são definidos os diferentes módulos presentes (já mencionados em cima). No ficheiro também são descritas as versões do Maven (compiler) utilizadas no projeto.

Quanto às dependências presentes no projeto, existem as dos diferentes módulos com o seu groupId e artifactId (o mesmo do projeto) e o Wildfly, que é o servidor utilizado no projeto, onde indica a versão utilizada, credenciais de acesso (no localhost).

Está presente também a dependência do Jakarta EE que é um conjunto de especificações que permite desenvolver "cloud Java Enterprise Applications". Através desta, são utilizados vários packages como "java.persistence" para conseguir gerir as persistências para "object/relational mapping", para ser possível a utilização de servlets ("javax.servlet") e de EJBs.

Nos diversos módulos criámos as páginas necessárias (também referidas anteriormente, em cada etapa, como é o caso das Classes Java no módulo JPA para a criação das entidades e as suas relações). Por fim, utilizámos o Docker para fazer "deploy" dos diferentes serviços (e respetivo software), para conseguirmos agrupar as diferentes packages no seu específico container (e no seu ambiente).

References

- Laranjeiro, N. and Amaral, F. (2021) *Jakarta EE in Practice*