Integração de Sistemas 2021/22

Assignment #2

Three-tier Programming with Object-Relational Mapping

Ana Beatriz Marques, 2018274233
Hugo Miguel Abreu, 2018275541

1. Introdução

Este projeto foi realizado em contexto da disciplina de Integração de Sistemas e pretende implementar uma web application para gerir uma empresa de autocarros. Este projeto foi realizado na Linguagem Java através de JEE (Jakarta EE), usando EJB (Enterprise JavaBeans) e JPA (Java Persistence API).

2. Presentation

2.1 Interface Manager

A interface manager consiste em 2 Menus: Login e Menu Principal.
Tal como pedido no enunciado, o registo de managers é realizado fora da interface, pelo que apenas é possível realizar login e nunca registar ou alterar informação do manager.
Iniciamos a Interface realizando um Login válido. Após o Login, entramos no Menu Principal onde é possível aceder a todas as funcionalidades para Managers:

- Create Trip: criar uma viagem com todos os parâmetros necessários;
- Delete Trip: eliminar uma viagem futura, devolver o valor do bilhete a todos os clientes e enviar email a informar a alteração;
- List Top 5: devolver os clientes com mais bilhetes comprados, independentemente se foram viagens já realizadas ou futuras;
- List Trips: esta funcionalidade pode ser um dos seguintes modos
    - listar viagens de um certo dia
    - listar viagens entre 2 datas
    - listar todas as viagens futuras
- List Users: apenas possível após o List Trips, em que é possível listar todos os utilizadores de uma certa viagem;

Cada funcionalidade relaciona-se com uma ou mais funções do EJB.

2.2 Interface User

Tal como na Interface Manager, iniciamos com o menu inicial: Login ou Registo.
No caso de Registo, verificamos a informação pedida e confirmamos que o email não está já registado, se o Registo for bem sucedido, o Login é feito automaticamente.
Após Login entramos no Menu Principal onde estão as funcionalidades de user:

- Buy Ticket: Listar as viagens futuras pela sua origem e destino e comprar um bilhete para a selecionada, apenas é possível comprar um bilhete por utilizador;
- Return Ticket: Listar as viagens do cliente e devolver o bilhete para a selecionada, recebendo o valor do bilhete de volta na wallet;
- List Trips: esta funcionalidade pode ser um dos seguintes modos
    - listar todas as viagens na qual o cliente comprou bilhete
    - listar viagens entre 2 datas
- Increment Wallet: adicionar dinheiro à wallet do cliente
- Change Info: alterar a informação pessoal do cliente, incluindo o email da conta
- Delete Account: eliminar a conta e tudo relacionado com a mesma, não é possível reaver o dinheiro da wallet pois apenas é possível adicionar à wallet e nunca remover

3. EJB

O EJB, Enterprise JavaBeans fornece unidades independente e reutilizáveis (beans) que podem ser usadas por outros métodos, realizando uma separação entre a as diferentes camadas de um programa, em especifico, javabeans faz parte da business layer, conectando a classes contidas na database com a interface.

3.1 EJB User

A classe EJB User permite, através do EJB User Remote, ser possível aceder/alterar informação de forma remota.
Contém as funções chamadas na interface do User que, consoante o desejado, realiza um query e retorna o resultado.

```java
//Login User
public Utilizador loginUser(String email, String password){
    try{
        TypedQuery<Utilizador> q = em.createQuery("SELECT DISTINCT u FROM Utilizador u WHERE u.email=:email", Utilizador.class)
            .setParameter("email",email);
        Utilizador u= q.getSingleResult();
        if(u.getPassword().equals(getHashedPassword(password))) return u;
        return null;
    } catch(NoResultException e) {
        return null;
    }
}
```

Podemos exemplificar a implementação de uma funcionalidade: Login do utilizador;

3.2 EJB Manager

Chamado pela Interface de Manager. Realiza as alterações necessárias na database pedidas e devolve o resultado sem o interpretar, diferenciando a *Business Layer* que é o EJB da *Presentation Layer* ,o JPA.

```java
//checks if email is already in, if it isnt we add the manager
public Manager registerManager(Manager manager) {

    TypedQuery<Manager> q = em.createQuery("SELECT DISTINCT m FROM Manager m WHERE m.email=:email", Manager.class)
            .setParameter("email",manager.getEmail());
    List<Manager> m= q.getResultList();
    if(m.size()==0){
        em.getTransaction().begin();
        manager.setPassword(manager.getHashedPassword());
        em.persist(manager);
        em.getTransaction().commit();
        return manager;
    }
    else{
        return null;
    }
}
```

Podemos exemplificar a implementação de uma funcionalidade: Registo de um Manager;

```java
@Schedule(hour="23", minute="59", second="0")
public void scheduledEmail(){
    List<Trip> trips = getRevenueTrips();

    Double count_money=0.0;
    int count_people=0;
    for(Trip t: trips){
        count_money += t.getOccupancy() * t.getPrice();
        count_people += t.getOccupancy();
    }
    String message;
    if (count_people>0) message="Earned "+ count_money + " euros from " + trips.size() +
                        " trips, with a total of "+ count_people + " tickets bought!\n";
    else{
        message="No one bought tickets today!\n";
    }

    TypedQuery<Manager> q = em.createQuery("SELECT DISTINCT m FROM Manager m", Manager.class);
    List<Manager> m= q.getResultList();
    if(m.size()<=0) return;
    /*
    EJBEmail email= new EJBEmail();
    for(Manager manager: m)  email.sendEmail(manager.getEmail(), "Bus company- Daily Revenue", message);
    */
    return;
```

```
private List<Trip> getRevenueTrips() {

    SimpleDateFormat formatter=new SimpleDateFormat("dd/MM/yyyy HH:mm");
    LocalDateTime now = LocalDateTime.now();
    String date = formatter.format(now);
    String [] date2 = date.split(" ");
    date2[1] = "00:00";
    String todayl = date2[0]+" "+date2[1];
    Date today = null;
    try {
        today = formatter.parse(todayl);
    } catch (ParseException e) {
        e.printStackTrace();
    }

    TypedQuery<Trip> q = em.createQuery("SELECT DISTINCT t FROM Trip t WHERE  t.time= :day AND t.time < now()  ORDER BY t.time", Trip.class)
            .setParameter("day", today);

    List<Trip> trips= q.getResultList();
    return trips;
}
```

Além das funcionalidades chamadas pela interface, têm uma função Schedule, ou seja, é chamada automaticamente na frequência pretendida, no nosso caso, diariamente ao final do dia.

3.3 EJB Email

```
@Stateless
@LocalBean
public class EJBEmail {

    @Resource(lookup = "java:jboss/mail/jeeapp") //change name of resource in wildfly
    private Session mailSession;
    private String from="buscompany@gmail.com";
    public EJBEmail() {
    }

    @Asynchronous
    public void sendEmail(String to, String subject, String content) {
        try
        {
            Message message = new MimeMessage(mailSession);
            message.setFrom(new InternetAddress(this.from));
            message.setRecipients(Message.RecipientType.TO,InternetAddress.parse(to));
            message.setSubject(subject);
            message.setText(content);
            Transport.send(message);
        }
        catch (MessagingException e)
        {
            System.out.println("Error sending mail : "+ e );
        }
    }

}
```

Este Bean permite enviar email através do Wildfly, apenas é chamado no EJB Manager nos seguintes métodos: *sheduledEmail*() e *DeleteTrip*().

4. Classes

4.1 Utilizador

```
@Entity
public class Utilizador implements Serializable
{

    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String password;
    private String address;
    private int phone;
    private String email;
    private double wallet;
    private boolean session;
```

A classe Utilizador representa qualquer cliente, pelo que é guardado toda a informação necessária para a compra de bilhetes nela. De forma a garantir a segurança dos utilizadores a password passa por uma função Hash.

4.2 Manager

```
@Entity
public class Manager extends Utilizador
{
    public Manager(){
        super();
    }

    public Manager(String name, String password, String address, int phone, String email){
        super(name, password, address,phone,email);
    }
```

O Manager é uma classe que estende o Utilizador, para determinar se o cliente tem ou não as permissões para aceder ao Manager Interface e receber emails diários do revenue. Isto não interfere na possibilidade de um Manager ser cliente da companhia o que implica que não existe nenhuma alteração entre o Manager e Utilizador.

4.3 Trip

```
@Entity
public class Trip implements Serializable {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String origin;
    private String destination;
    private Date time;
    private double price;
    private int capacity;
    private int occupancy;

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "trip")
    private List<Ticket> tickets;

    public Trip(){}
```

A classe Trip pode ser descrita nos parâmetros acima, e tal como indicado no enunciado, adicionamos extra parâmetros:

- capacity: número de lugares total do autocarro
- occupancy: número de lugares ocupados

Apesar de não serem necessários, consideramos a necessidade destes parâmetros na realidade de uma viagem.
Podemos também ver que cada Trip contém uma lista de tickets.

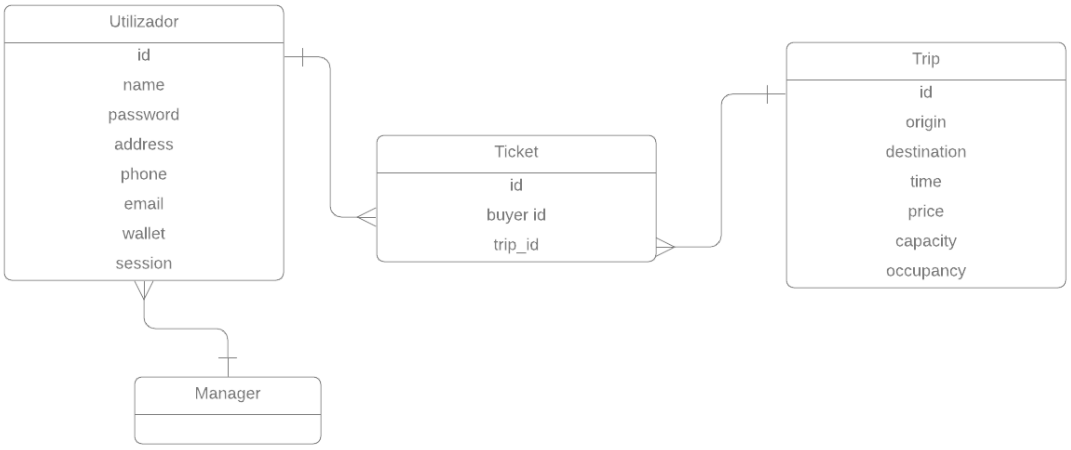
4.1 Ticket

```
@Entity
public class Ticket implements Serializable {
    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private Long buyer_id;

    @ManyToOne()
    @JoinColumn(name="trip_id", referencedColumnName = "id")
    private Trip trip;
```

Como previamente descrito, cada Trip corresponde a diversos tickets, pelo que, de forma a guardar corretamente na database, vamos buscar o id da Trip e guardamos o mesmo. Sempre que um bilhete é comprado pelo utilizador, guardamos o seu id.

5. ER



6. Package

No pom.xml temos primeiramente a declaração dos vários módulos (ejbs , jpa, ear, web).

```
<modules>
  <module>ejbs</module>
  <module>jpa</module>
  <module>ear</module>
  <module>web</module>
</modules>
```

Depois, temos as várias dependências necessárias para a execução do projeto, ou seja, os plugins do maven, jakartaee-api e também plugins do wildfly onde temos o username , password e porto de acesso à consola do mesmo.

```
    <dependency>
      <groupId>jakarta.platform</groupId>
      <artifactId>jakarta.jakartaee-api</artifactId>
      <version>8.0.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<build>
  <finalName>${project.artifactId}</finalName>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.wildfly.plugins</groupId>
        <artifactId>wildfly-maven-plugin</artifactId>
        <version>${wildfly-plugin-version}</version>
        <configuration>
          <skip>false</skip>
          <hostname>wildfly</hostname>
          <port>9990</port>
          <filename>${project.artifactId}.ear</filename>
          <username>admin</username>
          <password>admin#7rules</password>
        </configuration>
      </plugin>
```

Em relação à estrutura do projeto, temos 4 módulos diferentes.

Ejbs :

Contém os beans e as suas devidas interfaces remotas de forma às interfaces do manager e user terem acesso aos métodos remotamente.

Jpa:

Contém as estruturas de dados do projeto (Utilizador , Trip, Ticket e Manager).

Ear:

Contém as interfaces do utilizador e do manager , onde estes podem executar as diversas funcionalidades do projeto.

Web:

Não implementado.

Qualquer dúvida contactar-nos através de um dos seguintes emails:

anafm@student.dei.uc.pt

abreu@student.dei.uc.pt