Accessing a DB through an EJB layer

Davide Lissoni Mat.179878
21/10/2016

1.Introduction:

The project consists in writing two different main clients, one for manage user operation and the other one for administrator operations and a site in order to handle them.
The initial requirements for this project were to develop four different clients, but I preferred to merge these for simplify the usage of the applications, keeping in any case all the functionalities required.
The project essentially reproduces a Library management system used to buy books.
In particular the user through the **UserClient** can:

- register itself on the database;
- log into the system;
- visualize all the book available in the library
- get a cart;
- add one or more books to the user cart;
- visualize the cart and clear it;
- buy the books present in the cart;
-

Note that an user can run the cart operations even if is not logged into the system, but he/she cannot buy the content cart before the log in.

AdministratiorClient contains instead admin operations, that consist in:

- adding new books on the database;
- visualize the buying operation (which books has been sold and who is the customer).

Furthermore the instructions for this assignment specify that, the project must use stateful, stateless beans and entities.
I handle this request saving Account (username,password) and Book (title,price,buyer) informations in a database, with the help of EJB and two Entity class(Accont and Book). Account and book information are looked up in stateless EJBs, while the cart is looked up in a stateful EJB.
The motivations of these choice will be explained in the “Explanation” chapter.

2.Implementation:

2.1 Server side:

This part of the project consists in a Java Enterprise application, that will be deployed with WildFly (version 9.0.1.Final). The application, practically, is formed by three session beans (AccountManager, Bookmanager and Cart manager) and two entities (Account and Book).
The database access occurs via AccontManager and BookManager stateless bean that provides methods used to get and query their entity attributes based on their id or other parameters.

```
import javax.persistence.Query;

@Stateless
public class AccountManager implements AccountManagerRemote {

    @PersistenceContext
    private EntityManager manager;


    @Override
    public void registerAccount(String usr, String pwd) {
                Account account = new Account(usr,pwd);
                manager.persist(account);
    }
```

Figure 1

In particular **AccountManager(**Figure 1**)**, besides having getter methods, contains:

- **registerAccount(usernamme,password)**: create a new instance Account and create a new record in the database for the entity by a *manager.persist(account)*.
- **login(**Figure 2**)**: query the database looking for account that have the username and password taken as parameter and return the account id of the respective user.

```
@Override
public int login(String usr,String pwd){
  int id;
Query q= manager.createQuery("SELECT a FROM Account a WHERE a.username = :user AND a.pwd = :pass");
    q.setParameter("user", usr);
    q.setParameter("pass", pwd);
    List <Account> res= q.getResultList();
      if (res.size()==0)id=0;
      else id=res.get(0).getAccountInfo().getId();


return id;
```

Figure 2

BookManager beans is looked up both in the UserClient and in the AdministratorClient and contains the following methods:

- **getBookInfo**: create a bookInfo instance containing the book attributes;
- **addBook(pw admin)**(Figure 3):Check if the password admin passed is correct, then create a new Book instance and add a new record in the database for the entity by a *manager.persist(book)*.The password admin is a constant saved on the bean and currently is setted “pw”;
- **listbook()**(Figure 4): query the database and get all the books available in the library(just books that have not been sold)and return a *List<String>* of formatted string containing the books informations;
- **listBuying(pw admin)**: Check if the password admin passed is correct and then query the database in order to return a *List<String>* of formatted string containing the buying informations(books and relative customers).

```
@Override
public boolean addBook(String name, int price, String pw) {
 if(pw.equals(ADMIN_PASSWORD)){
    Book book= new Book(name,price);
  manager.persist(book);
 return true;}else{return false;}
}
```

Figure 3

```
   @Override
   public List <String> listBook() {
Query q= manager.createQuery("SELECT b FROM Book b WHERE b.buyer = ?1");
        q.setParameter(1, 0);
        List <Book> res= q.getResultList();
        List <String> books= new ArrayList<>();

 for (int i=0; i<res.size(); i++){
 books.add("id: "+res.get(i).getBookInfo().getId()+" name: "+res.get(i).getBookInfo().getName()+" price: "+res.get(i).getBookInfo().getPrice());
 }
   return books;}
```

Figure 4

The **CartManager**(Figure 5) Stateful bean contains as data a *<Book> List*, used to save the books added to the cart by the user and a *buyerid* as account reference. This bean provides two constructor, one for the logged user, and one for anonymous " user". In this way, an unlogged user can get a cart and make operation on it. If the same user, after have got a cart and have performed some operation on it, decide to log in in the same session, his cart will be retrieved keeping its state, letting him/her to buy the cart content.

```
@Stateful
public class CartBean implements CartRemote {

    int customerId=-1;
    List<Book> contents;
    @PersistenceContext
    private EntityManager manager;
```

Figure 5

The beans contains therefore also the buy method which update the database by setting in the Book entity the idBuyer, that will be the account id of the user that bought the book(Figure 6).

```
@Override
public void buy(){
for (Book book: contents){

book.setBuyer(customerId);
manager.merge(book);
}
contents.clear();
}
```

Figure 6

Account entity contains as attributes the automatic generate accountId(Figure 7),the username and the password. The entity has also a method to get the account information.

```
@Entity
public class Account implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int accountId;
    private String username;
    private String pwd;
```

Figure 7

Book entity contains an automatic generate accountId as well,the title,the price and the id buyer, setted to "0" when a new book entity is created. The entity has also a method to get the book information.
For these entities has not been use particular annotations.

The **persistence file(***persistence.xml***)** is composed by just a single unit(Figure 8).

```
<persistence-unit name="manager">
      <jta-data-source>java:/LibraryDS</jta-data-source>
      <class>entities.Account</class>
      <class>entites.Book</class>
      <properties>
        <property name="hibernate.hbm2ddl.auto" value="update"/>
      </properties>
  </persistence-unit>
</persistence>
```

Figure 8

2.2 UserClient

The user can run all the features mentioned in the introduction chapter for this client.
The client interact with the user through command line using simple based command that will be listed during the running of the application.
The program get initially all the beans (AccountManager,BookManager and CartManager) (Figure 9) and start a cycle where ask to the user what to do. At this point the user can choose:

- **to register a new account**: in this case the relative AccountManager method will be called;
- **to log in**: also the log in use its AccountManager method;
- **to get a cart without log**: in this case will be used *CartManager.initialize()*, after that will be displayed the list of the available books and will allows the user to add books to the cart or clear it.

When a user do the log in the system perform in the following way:

1. Check if the user got a cart before the log in. If yes just set the idaccount on the cart bean, if not instead initialize a new cart using *CartManager.initialize(account id)*;
2. Start a cycle used to manage the cart operations by the methods displayed in the figure 9

```
case "1":addBook(cm); break;
case"2":showCart(cm);break;
case"3":cm.buy();break;
case"4":cm.leave();break;
case "5": cycleInside=false;  break;
default: System.out.println("Invalid selcetion:\n");id=0;break;
```

Figure 9

2.3 AdministratorClient

Also this client has not user interface but interact with simple command in command line.
This client, interacting with just the Book entity, needs to retrieve only the BookManager bean.
Once got the bean the system ask the user if add a new book or list the buying operation and, in both cases, ask to insert the password admin. The application will called the bean methods according to the user choice.

3.Explanation:

3.1 Entities

This business application needs to save some persistent data and to maintain them in a database. in particular Account and Book information. The Entity Beans enable automated management of certain aspects of access and, in general to data management. So for this project has been implemented two entities (**Account**, **Book**)identified by a auto-generated primary key and with their relative attributes.

3.2 Session beans

The two clients call three different session beans in order to invoke their methods, and for this application I decide to implement the following session beans:

- **AccountManager**: this session beans, as already explained in the implementation chapter, implements a method used to get user information, the *login(username,pwd)* and the *registerAccount(username,pwd)*. All this method don't need to remember the state of the previous calls. For this motivation AccountManager has been implemented as **stateless session bean.**
- **BookManager** is used instead for methods call in order to manage the Book entity. For the same motivation of AccountManager, BookManager is a **stateless session bean**.
- **CartManager** implements the methods use to manage the Cart (i.e. *addBook(title,price) buy(id book)* and *clear()*). Since there is the need to keep the values of the data present in the session bean for every call, for instance the list of the books added to the cart, CartManager is a **stateful session bean**.The stateful session beans in fact guarantee the maintenance of the communication status, thanks to the EJB container that shall ensure that the same session bean instance manage different bean method invocation called by the same client.

4.Deployment:

The first step is starting the DBMS, where the two clients need to access in order to retrieve book

and account information and to configure the JDBC Datasource in the Wildfly console. For the development of this project I chose to use derby(version. 10,12,1,1), following in this way the professor guide.
Derby will run on the derby default port 1527 but it is also possible to change it if necessary.
In order to start Derby go in the derby located folder and type the command:
java -jar lib/derbyrun.jar server start &

In order to configure the JDBC Datasource in Wildfly, open the administrator console and upload the derby JDBC driver (derbyclient.jar) on the deployment tab. After that, add a datasource calling it “LibraryDS” adding, as driver, the one just uploaded. If the connection test succeed the Datasource configuration is completed.

The next step is to deploy on Wildfly the Enterprise Application LibraryServerSession.
In order to do that, copy LibraryServerSession.ear in the deployments folder located on the Wildfly installation directory and then restart Wildfly. Your output should look like(Figure 10,11):

```
17:07:55,021 INFO  [org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeplo
ymentUnitProcessor] (MSC service thread 1-7) JNDI bindings for session bean name
d BookManager in deployment unit subdeployment "LibraryServerSession-ejb.jar" of
 deployment "LibraryServerSession.ear" are as follows:

        java:global/LibraryServerSession/LibraryServerSession-ejb/BookManager!be
ans.BookManagerRemote
        java:app/LibraryServerSession-ejb/BookManager!beans.BookManagerRemote
        java:module/BookManager!beans.BookManagerRemote
        java:jboss/exported/LibraryServerSession/LibraryServerSession-ejb/BookMa
nager!beans.BookManagerRemote
        java:global/LibraryServerSession/LibraryServerSession-ejb/BookManager
        java:app/LibraryServerSession-ejb/BookManager
        java:module/BookManager
```

Figure10

```
17:07:57,806 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 1) WFLYSRV
0010: Deployed "LibraryServerSession.ear" (runtime-name : "LibraryServerSession.
ear")
17:10:02,322 INFO  [org.jboss.ejb.client] (pool-1-thread-1) JBoss EJB Client ver
sion 2.1.1.Final
```

Figure 11

Now both the clients can be run using the following commands:

UserClient: java -jar UserClient.jar (Figure 12)
AdministratorClient: java -jar AdministatorClient.jar (Figure 13)

The user will interact by command line.
The interaction of the application chosen will be explained during its running.

```
INFO: EJBCLIENT000013: Successful version handshake completed for receiver conte
xt EJBReceiverContext{clientContext=org.jboss.ejb.client.EJBClientContext@7cd62f
43, receiver=Remoting connection EJB receiver [connection=Remoting connection <3
566b2e5>,channel=jboss.ejb,nodename=davide-hp-pavilion-dv6-notebook-pc]} on chan
nel Channel ID f595abd4 (outbound) of Remoting connection 5700d6b1 to localhost/
127.0.0.1:8080
ott 21, 2016 5:20:41 PM org.jboss.ejb.client.EJBClient <clinit>
INFO: JBoss EJB Client version 2.1.1.Final
Press:
1 to register a new user
2 to log in (if you already add some books in a cart the card will be retrived)
3 if you want to get a new cart without log in
4 if you want to quit
```

Figure 12

```
INFO: EJBCLIENT000013: Successful version handshake completed for receiver conte
xt EJBReceiverContext{clientContext=org.jboss.ejb.client.EJBClientContext@604ed9
f0, receiver=Remoting connection EJB receiver [connection=Remoting connection <4
65c43c2>,channel=jboss.ejb,nodename=davide-hp-pavilion-dv6-notebook-pc]} on chan
nel Channel ID 92edd1cc (outbound) of Remoting connection 3fee9989 to localhost/
127.0.0.1:8080
ott 21, 2016 5:21:20 PM org.jboss.ejb.client.EJBClient <clinit>
INFO: JBoss EJB Client version 2.1.1.Final
Press:
1 if you want to add a new book
2 if you want to list the buying operations
3 if you want to quit
```

Figure 13

5.Comments and notes

Since it was very tricky and it required a lot of time to configure and make available all the staff for the development of this project, especially on how to configure the JDBC datasource on Wildfly, the applications have no user interfaces. This can allow some command errors. Please follow strictly the indication printed during the run of the two clients.