23 Library management with EJB and JPA

Bărcan Florin

Bîrsan Diana

Contents

[Theoretical aspects 3](#_Toc95667471)

[EJB 3](#_Toc95667472)

[JPA 3](#_Toc95667473)

[JSP 3](#_Toc95667474)

[Wildfly application container 3](#_Toc95667475)

[Project configuration 4](#_Toc95667476)

[Application architecture 5](#_Toc95667477)

[Database 6](#_Toc95667478)

[Login Mechanism 7](#_Toc95667479)

[Deploying the application 7](#_Toc95667480)

[Bibliography 8](#_Toc95667481)

# Theoretical aspects

## EJB

**Enterprise Java Beans** or EJB are server-side software components that encapsulate the business logic of an application. An EJB web container provides a runtime environment for web related software components, including computer security, Java servlet lifecycle management, transaction processing, and other web services.

## JPA

**Java Persistence API** or JPA is a set of concepts that is concerned with persistence inside a Java application, which loosely means any mechanism by which Java objects outlive the application process that created them. Not all Java objects need to be persisted, but most applications persist key business objects. The JPA specification lets you define which objects should be persisted, and how those objects should be persisted in your Java applications.

## JSP

**Java Server Pages** or JSP is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. This tutorial will teach you how to use Java Server Pages to develop your web applications in simple and easy steps.

## Wildfly application container

**WildFly** is a flexible, lightweight, managed application runtime that helps you build amazing applications.

# Project configuration

First, we had to connect our database to the wildfly server.

For the database we have used mysql and apache.

We needed to define a JDBC driver, along with a Datasource for the Wildfly server, and include them in the persistence file of our project, under /resources/META-INF/persistence.xml. Later, we will also declare our entities there, in order to be able to use them.

The configuration for the **standalone.xml** file of the wildfly server looks like this:

<xa-datasource jndi-name="java:jboss/datasources/**MySqlDS**" pool-name="MySqlDS">

<xa-datasource-property name="ServerName">

localhost

</xa-datasource-property>

<xa-datasource-property name="databaseName">

la\_biblioteca

</xa-datasource-property>

<xa-datasource-property name="port">

3306

</xa-datasource-property>

<xa-datasource-class>com.mysql.cj.jdbc.MysqlXADataSource</xa-datasource-class>

**<driver>mysql</driver>**

<security>

<user-name>root</user-name>

</security>

<validation>

<valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>

<validate-on-match>true</validate-on-match>

<exception-sorter class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>

</validation>

</xa-datasource>

The persistence file looks like this:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<persistence xmlns="https://jakarta.ee/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence\_3\_0.xsd"

version="3.0">

<persistence-unit name="ejb">

<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

<jta-data-source>java:jboss/datasources/**MySqlDS**</jta-data-source>

<class>entities.User</class>

<class>entities.Carte</class>

<class>entities.Inchirieri</class>

<properties>

<property name="hibernate.dialect" value="org.hibernate.dialect.MariaDBDialect"/>

<property name="hibernate.show\_sql" value="true"/>

</properties>

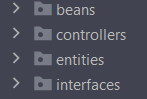
</persistence-unit>

</persistence>

# Application architecture

The backend side of our project is using EJB and JPA for managing 3 tables from a database.

This way, the backend side of the project is structured like this:

The **beans** folder is holding the beans logic, that follow their respective interfaces from the **interfaces** folder and entities are the base entities, declared accordingly to their respective table.

If we take the **user\_bib** table for example, we will have the following flow for entity->interface->bean

**ᐯ entities/User**

@Entity

@Table(name="user\_bib")

public class User implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int uID;

private String nume;

....

}

**ᐯ interfaces/IUser**

public interface IUser {  
 User add(User user);

...

}

**ᐯ beans/UserBean**

@Stateless

@Remote(IUser.class)

public class UserBean implements IUser {

@PersistenceContext(unitName = "ejb")

private EntityManager entityManager;

@Override

public User add(User user) {

entityManager.persist(user);

return user;

}

....

}

Next, we have the **controllers** package, which holds our Dispatcher object, that controls requests and responses and manages our JSP pages and the LoginController.

# Database

For this project we have used a mysql database, named **la\_biblioteca**, which contains three tables:

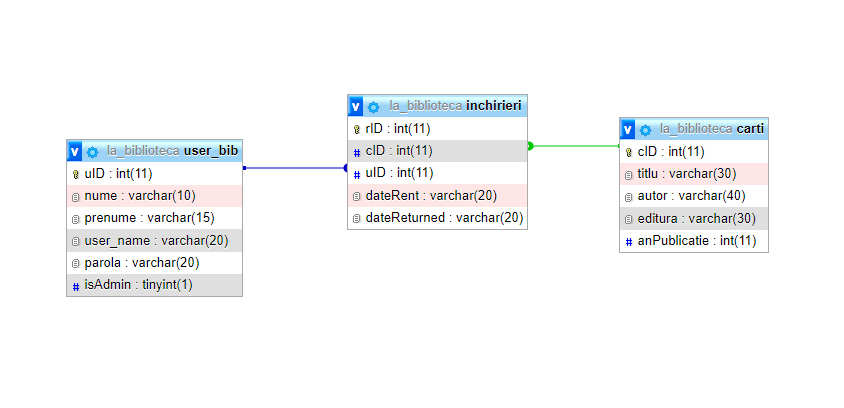
1. The **user\_bib** table represents the users of the application, which can be of two types
   1. Admins – they can manage the list of books by adding a new book, deleting or editing an existent book
   2. Normal users – they can rent or return a book; so they will be able to see a list of books that are available for renting, along with a list of books they have rented
      1. a user cannot rent a book they have not returned and cannot rent more than 5 books at a time

A user has multiple properties: **nume**, **prenume**, **user\_name** and **parola**, all strings, and a boolean field that represents the admin rights of the user.

1. The **carti** table is responsible for our books

A book has the following fields: **titlu**, **autor**, **editura**, all strings and an integer field, **anPublicatie**.

1. The **inchirieri** table represents the intermediary table for the many-to-many relationship a normal user can have with books. ( A user can rent more books, and a book can be rented by many users). This table holds information about the rentDate and returnedDate (strings), as long with the book and user’s id.



# Login Mechanism

# Deploying the application

We have chosen Wildfly as the container for deploying our application. After configuring both the project and the Jboss Server, the application can be deployed by being placed in the /standalone/deployments folder of the server, or by using the console provided by it.

It can be accessed in the browser at: <http://127.0.0.1:8080/wildfly-1.0-SNAPSHOT/dispatcher>

# Bibliography

<https://docs.wildfly.org/>

<https://www.tutorialspoint.com/jsp/index.htm>

<https://www.tutorialspoint.com/ejb/index.htm>

<https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html>

Lecture notes