

EjbJpa

Despre proiect

Acest proiect utilizează Enterprise JavaBeans (EJB) și Java Persistence API (JPA), împreună cu Servlet-uri. Proiectul include **un server**, responsabil pentru gestionarea a două tabele în baza de date, și **doi clienți**: unul care interacționează cu serverul prin Java Naming and Directory Interface (JNDI), iar celălalt folosind injectarea EJB.

Scopul principal al proiectului este de a gestiona o colecție de filme, permițând utilizatorilor să adauge filme în biblioteca lor personală.

Aplicația este ambalată sub forma unei arhive WAR (Web Application Archive), care poate fi desfășurată pe serverele de aplicații GlassFish și WildFly. Există diferențe minore între codul sursă și configurarea fișierului de persistență pentru fiecare server de aplicații.


Datorită acestor diferențe, s-au creat două versiuni separate ale proiectului, câte una pentru fiecare server de aplicații. Aceste versiuni conțin codul sursă specific pentru partea de server și client, utilizând bean-uri EJB prin injectare. Diferențele sunt după cum urmează:

- Pentru WildFly:
 - o Sursa de date (datasource) în fișierul de persistență *resources/META-INF/persistence.xml* este specificată ca `<jta-data-source>java:jboss/datasources/MySQLDS</jta-data-source>`.
 - o Variabila care reprezintă ID-ul entităților mapate la tabele în baza de date este adnotată cu `@GeneratedValue`.
- Pentru GlassFish:
 - o Sursa de date în fișierul de persistență *resources/META-INF/persistence.xml* este specificată ca `<jta-data-source>jdbc/mysql</jta-data-source>`.
 - o Variabila care reprezintă ID-ul entităților este adnotată cu `@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQMYCLASSID")`.


Există 3 pagini:

- O pagină pentru a vedea toată colecția de filme
 - o <http://localhost:8080/ejb-jpa-1.0/>
 - o <http://localhost:8080/ejb-jpa-1.0/view-movies>
- O pagină în care user-ul trebuie să își treacă numele pentru a cumpăra o copie de film
 - o <http://localhost:8080/ejb-jpa-1.0/purchase-movie?movieID=1>
- O pagină în care user-ul își poate vedea toate achizițiile
 - o <http://localhost:8080/ejb-jpa-1.0/show-user-movies?username=username1234>


Cele 3 pagini pot fi văzute mai jos:



White Chicks
Genre: Comedy
Release Year: 2004
[Purchase](#)



Life Is Beautiful
Genre: Comedy-Drama
Release Year: 1997
[Purchase](#)



Terminator
Genre: Action
Release Year: 1984
[Purchase](#)

Purchase - White Chicks

Name

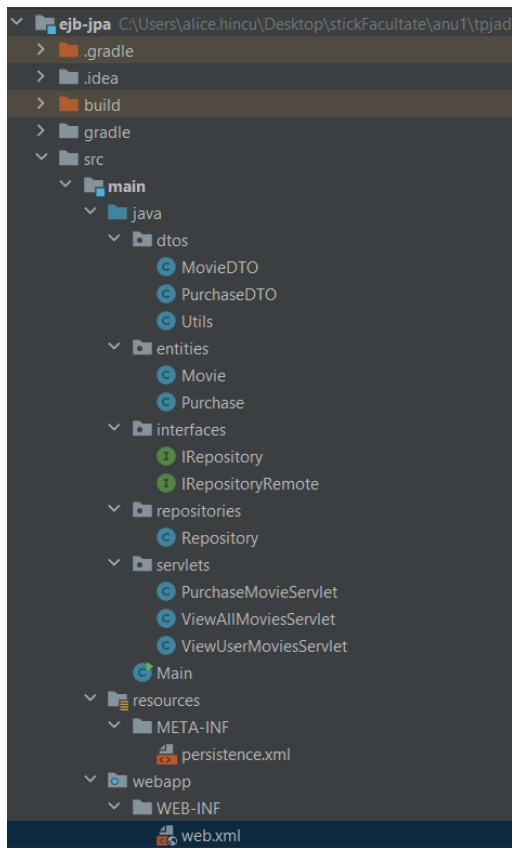
[Purchase Movie](#)

username1234's Movie Collection

White Chicks, purchased on 18-01-2024

[Return Home](#)

Structura aplicației



Entitățile relevante, care sunt persistate în baza de date MySQL și între care există o relație de one-to-many, au fost modelate în felul următor:

Clasa **Movie**:

- Long **movieID**: id-ul entității
- String **title**: titlul filmului
- int **releaseYear**: anul de lansare al filmului
- String **genre**: genul filmului
- String **imageUrl**: posterul filmului
- Collection<Order> **purchases**: lista de comenzi a userilor din care jocul face parte; acest câmp este adnotat cu `@OneToMany(mappedBy = "movie", cascade = CascadeType.ALL, fetch = FetchType.EAGER)`

Clasa **Purchase**:

- Long **purchaseID**: id-ul entității
- String **buyerName**: numele utilizatorului care a achiziționat filmul
- String **purchaseDate**: data la care achiziția a luat loc
- Movie **movie**: filmul achiziționat; acest câmp este adnotat cu `@ManyToOne și @JoinColumn(name = "movieID")`

Se observă că un utilizator poate achiziționa mai multe filme printr-un proces de „cumpărare” (purchase), în timp ce fiecare proces de cumpărare este asociat cu un singur film, creând astfel o relație de tipul unu-la-mai-multe (one-to-many) între filme și cumpărături.

Pentru aceste entități, am creat Data Transfer Objects (DTO-uri), care includ câmpuri specifice fiecărei entități. Aceste DTO-uri sunt utilizate pentru a facilita comunicarea cu clienții prin intermediul JNDI.

În materie de bean-uri EJB, am definit bean-ul *Repository* al cărui scop este să conțină logica de business a aplicației, și care implementează interfețele *IRepository* și *IRepositoryRemote*.

Interfața *IRepository* specifică toate metodele esențiale pentru aplicație – de exemplu, metode pentru crearea și căutarea unei entități specifice, precum și pentru căutarea tuturor entităților de un anumit tip. Aceste metode au fost elaborate pentru fiecare entitate prezentă în proiect. Pe de altă parte, interfața *IRepositoryRemote* conține metode similare, însă acestea sunt special concepute pentru a fi invocate de clienți JNDI la distanță, având un nume și un scop adaptat acestui context.

Servlet-uri:

- **ViewAllMoviesServlet** - Acest servlet este punctul de intrare în aplicație, prezentând o pagină HTML care afișează colecția de filme disponibile. De asemenea, oferă posibilitatea utilizatorilor de a efectua cumpărături pentru filmele dorite.
- **PurchaseMovieServlet** – Rolul acestui servlet este de a oferi o pagină HTML cu un formular prin care utilizatorii pot introduce numele pentru a efectua o achiziție. În plus, gestionează solicitările de cumpărare, înregistrându-le în baza de date.
- **ViewUserMoviesServlet** – Acest servlet furnizează utilizatorilor detalii despre colecția personală de filme, permițându-le să vizualizeze achizițiile efectuate.

Este evident că fiecare servlet necesită interacțiunea cu baza de date pentru a îndeplini funcțiile specificate. Interfața cu baza de date este gestionată prin intermediul bean-ului EJB *Repository*, care încapsulează logica de business a aplicației. Acest bean este injectat în servlet-uri utilizând mecanismul de injectare EJB, astfel:

@EJB

private IRepository repo;

Structura aplicației JNDI

Am dezvoltat încă două proiecte care cuprind clienți JNDI specifici pentru WildFly și GlassFish. Acești clienți sunt setați să se conecteze la aplicația desfășurată pe serverul de aplicații (AS) corespunzător, folosind un context configurat în mod particular. După stabilirea

conexiunii, ei localizează interfața remote a bean-ului EJB de pe server utilizând denumirea JNDI specificată, astfel:

- Pentru WildFly, folosim: *ejb:/ejb-jpa-1.0/Repository!interfaces.IRepositoryRemote*
- Pentru GlassFish, utilizăm: *java:global/ejb-jpa-1.0/Repository!interfaces.IRepositoryRemote*

Acești clienți au o structură simplă după inițializarea instanței EJB remote. Scopul lor este să execute operațiuni de bază, cum ar fi inserarea de noi filme în sistem (obiecte de tip Movie), afișarea tuturor înregistrărilor din baza de date și alte funcționalități similare.

Instalarea și configurarea AS-urilor, SGBD-urilor și a mediului de dezvoltare

Proiectul a fost realizat pe o mașină ce rulează sistemul de operare Windows 10 64bit.

Pentru realizarea acestui proiect s-au folosit următoarele:

- Gradle 8.0: <https://gradle.org/releases/>
- JDK 1.8.0_392 temurin: <https://adoptium.net/temurin/releases/?version=8&os=windows>
- GlassFish 5.1.0 Full Profile: <https://projects.eclipse.org/projects/ee4j.glassfish/downloads>
- WildFly 20.0.1 Final: <https://download.jboss.org/wildfly/20.0.1.Final/wildfly-20.0.1.Final.zip>
- MySQL Connector Java 5.1.48: <https://cdn.mysql.com/archives/mysql-connector-java-5.1/mysql-connector-java-5.1.48.zip>
- MySQL Community 8.0.36: <https://dev.mysql.com/downloads/installer/>

Instalarea AS-urilor WildFly și GlassFish constă în extragerea acestora din arhive pe drive-ul local. Pentru a configura SGBD-ul MySQL în aceste AS-uri este necesar jar-ul pentru connector-ul de MySQL, care poate fi obținut de pe site-ul oficial menționat mai sus.

Configurarea SGBD-ului MySQL în GlassFish

- Se copiază jar-ul connector-ului de MySQL în %GLASSFISH_HOME%\glassfish\lib
- Se pornește consola AS-ului folosind fișierul %GLASSFISH_HOME%\bin\asadmin.bat
- Se pornește AS-ul din consola deschisă anterior folosind comanda *start-domain*
- Se execută următoarele comenzi, în ordine:

- *create-jdbc-connection-pool -restype javax.sql.DataSource --datasourceclassname com.mysql.jdbc.jdbc2.optional.MysqlDataSource --property "user=root:password=password:url=jdbc\:mysql\://localhost\:3306/ejb_db_gf:useSSL=false" MySqlPool*
- *create-jdbc-resource --connectionpoolid MySqlPool jdbc/mysql*
- La primul deploy al aplicației, se va folosi flag-ul *--createtables=true*, similar cu următoarea comandă: *deploy --createtables=true "PATH_TO_WAR"*
- **DACĂ SUNT ERORI CU PORTURILE 4848 SI 8080:**
 - *netstat -ano | findstr :<PORT>*
 - *taskkill /PID <PID> /F*
- **DACĂ SUNT ERORI CU PORTUL 3306 (nu merge conexiunea la baze de date):**
 - *Mysql Server > Reconfigure*

Configurarea SGBD-ului MySQL în WildFly

- Se copiază jar-ul connector-ului de MySQL în %WILDFLY_HOME%\bin
- Se pornește AS-ul folosind comanda %WILDFLY_HOME%\bin\standalone.bat
- Se pornește utilitarul jboss-cli aflat în locația %WILDFLY_HOME%\bin\jboss-cli.bat

În utilitarul jboss-cli se execută următoarele comenzi, în ordine:

- *connect*
- *module add --name=com.mysql --resources=mysql-connector-java-5.1.48.jar --dependencies=javax.api,javax.transaction.api*
- */subsystem=datasources/jdbc-driver=mysql: add(driver-name=mysql,driver-module-name=com.mysql,driver-xa-datasource-classname=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource)*
- *data-source add --name=MySqlDS --driver-name=mysql --jndi-name=java:jboss/datasources/MySqlDS --connection-url=jdbc:mysql://localhost:3306/ejb_db_gf --user-name=root --password=password --enabled=true*

Configurarea SGBD-ului MySQL în WildFly

Pentru a face deploy server-ului este necesară obținerea unei arhive WAR care poate fi obținută prin executarea comenzii `gradle clean build`. War-ul generat se va afla în locația `%PROJECT_ROOT%\build\libs`.

Pentru a face deploy arhivei war pe AS-ul GlassFish este necesară executarea comenzii `%GLASSFISH_HOME%\bin\asadmin.bat deploy "path_to_war"` iar pentru AS-ul WildFly este necesară copierea war-ului în `%WILDFLY_HOME%\standalone\deployments`. Odată ce s-a făcut deploy war-ului, aplicația poate fi accesată la adresa `localhost:8080/jar_name`.

Executarea clienților JNDI se poate realiza fie direct din IDE, fie prin generarea unui jar executabil, urmând a rula jar-ul generat. Un lucru ce merită menționat este că pentru rularea clientului JNDI pentru WildFly a fost necesară compilarea cu JDK 11.