# Documentație

## Implementare

În modulul Client

Avem o clasa StartClient care are o functie numita **public static int generateTicketNumber (int min, int max);** care genereaza un numar random intre min si max date si functie main in care se vor face vanzarile si testarea, daca s-a conectat cu success la server.

In package-ul resources cu ajutorul fisierului spring-client.xml clientul se conecteaza la server, daca acesta este activ.

În modulul Server

Avem clasa StartSever care are doar o singura functie, functia main care il porneste folosindu-se de spring-server.xml unde are loc conexiunea db – repo si repo – service si este activ timp de 2 minute. Si avem mai multe package-uri model, repo si service.

In package-ul model avem entitatile care vor fi folosite in aceasta aplicatie:

Entity -> clasa care implementeaza Serializable si are un id de tip int si e extinsa de restul claselor de entitati.

Sala -> care are un nr\_locuri de tip int.

Spectacol -> care are o data\_spectacol de tip LocalDate, un titlu de tip String, pret\_bilet si sala\_id de tip int si o lista de int-uri care reprezinta nr de locuri\_vandute.

Vanzare -> care are o data\_vanzare de tip LocalDate, nr\_locuri\_vandute si spectacol\_id de tip int si o lista de int-uri care reprezinta nr de locuri\_vandute.

In package-ul repo avem interfata extinsa de interfetele tuturor entitatilor avand o functie de adaugare si una de returnarea a tuturor entitatilor.

Interfata ISala care nu are metode in plus.

Interfata ISpectacol care are in plus metoda **List<Spectacol> findAllBySalaId(int salaId).**

Interfata IVanzare care are in plus metoda

**List<Vanzare> findAllBySpectacolId(int spectacolId).**

In package-ul database avem clasa JdbcUtils care face conexiunea cu baza de date si clasele Sala/Spectacol/VanzareDBRepository care implementeaza interfatele de mai sus.

In package-ul service avem interfata Iservice care defineste metodele care vor fi implementate si clasa Service care le implementeaza.

Metodele sunt :

**public List<Spectacol> getAllSpectacolBySalaId(int salaId);**

**public List<Vanzare> getAllVanzareBySpectacolId(int spectacolId);**

**public List<Sala> getAllSala();**

**public List<Spectacol> getAllSpectacol();**

**public List<Vanzare> getAllVanzare();**

**public int getSoldTotal();**

**public List<Integer> getLocuriLibere(int spectacol\_id);**

**public boolean checkLocuriLibere(int spectacol\_id);**

**public void addVanzare(Vanzare v);**

**public boolean checkSoldTotal();**

Tot in acest package avem si clasa de exceptii MyServerException. Exceptiile de acest tip sunt aruncate de server.

In baza de date avem 4 tabele in care se mentin entitatile, in tabela sala avem salile si nr de locuri pentru fiecare, in tabela spectacol mentinem atributele entitatii Spectacol, in tabela Vanzare avem id-ul , data\_vanzarii si id-ul spectacolului, iar in tabela vanzareLocuri avem id-ul vanzarii si nr de locuri cumparate.

Cand se face o vanzare se introduce atat in tabela vanzare cat si in cea de vanzarelocuri, iar cand se cauta toate vanzarile sau vanzarile dupa un spectacol id se interogheaza tot aceste 2 tabele.

## Rulare si testare

* Se porneste server-ul care va rula timp de 2 minute, lucru care este realizat cu ajutorul unui Timer din **java.util.Timer.**
* Cat timp serverul e activ, acesta face cu ajutorul unui thread separat si a unui timer o verificare odata la doua secunde a locurilor vandute prin verificarea corespondentei corecte intre locurile libere si vanzarile facute, sumele incasate per vanzare si soldul total. Rezultatele verificarilor sunt afisate in fisierul “verificari.txt “.
* Clasa service creeaza un **ExecutorService** din **java.util.concurrent.Executor.**

Acesta ve crea un Thread-Pool de NTHREADS thread-uri care vor executa task-urile (vanzarea de bilete si verificarea).

* Dupa ce server-ul a pornit, mai multi clienti se pot conecta la acesta pe acelasi port. Conexiunea se face folosind spring, clientul avand o referinta la bean-ul de server.
* Clientul, daca s-a conectat cu success, afiseaza **"S-a facut conexiunea la server!"**, iar daca nu s-a reusit conectarea va afisa **"Server-ul nu mai este activ!"**.
* Clientul foloseste 1 timer care face o vanzare la fiecare 2 sec, daca se poate realiza acest lucru.
* Clientul se foloseste de o functie

**public static int generateTicketNumber (int min, int max)**, functie folosita ca sa aleaga random id-ul spectacolului de la care va fi vanzarea si numarul de locuri cumparate (daca este mai mare ca nr de locuri libere ii dam valoarea de locuri libere). Cand nr de locuri libere la toate sepctacolele = 0 => timer-ul se va inchide dupa afisarea mesajului **"Nu mai sunt locuri**

**libere la niciun spectacol!".**

* Task-urile sunt executate de functii asincrone de tip **Future**<?>
* Daca clientul incearca sa cumpere niste locuri deja cumparate serverul va arunca o **MyServerException** al carei mesaj va fi afisat pe ecranul clientului.
* Datele de test:

Nr\_locuri =100; 3 spectacole (S1, S2, S3)

S1 pret\_bilet=100;

S2 pret\_bilet=200;

S3 pret\_bilet=150;

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |