<Aplicatie pentru gestionarea spectacolelor Operei Nationale Cluj-Napoca>

Student: Man Silvana

**Group: 30234**

Table of Contents

[1. Requirements Analysis 3](#_Toc512511305)

[1.1 Assignment Specification 3](#_Toc512511306)

[1.2 Functional Requirements 3](#_Toc512511307)

[1.3 Non-functional Requirements 3](#_Toc512511308)

[2. Use-Case Model 4](#_Toc512511309)

[3. System Architectural Design 4](#_Toc512511310)

[3.1 Architectural Pattern Description 4](#_Toc512511311)

[3.2 Diagrams 5](#_Toc512511312)

[4. UML Sequence Diagrams 6](#_Toc512511313)

[5. Class Design 6](#_Toc512511314)

[5.1 Design Patterns Description 6](#_Toc512511315)

[5.2 UML Class Diagram 7](#_Toc512511316)

[6. Data Model 8](#_Toc512511317)

[7. Bibliography 8](#_Toc512511318)

# 1. Requirements Analysis

## Assignment Specification

Opera Nationala din Cluj are nevoie de o aplicație pentru gestiunea spectacolelor si a biletelor care se pun in vânzare pentru spectacole. Aplicatia trebuie sa aiba doua tipuri de utilizatori: casieri si administrator. Accesul se face pe baza de login.

## Functional Requirements

Administratorii vor putea sa gestioneze lista de spectacole (operații CRUD). Pentru fiecare spectacol se vor salva in baza de date: genul (opera, opereta, balet), titlul, regia, distribuția, data premierei, număr de bilete care se pun in vânzare.

Administratorii vor putea sa creeze conturi pentru casieri (nume, username, parola).

Casierul va putea după login sa adauge in sistem bilete care s-au cumpărat pentru un anumit spectacol. Fiecare bilet va contine: spectacolul, rând, număr.

Sistemul va verifica sa nu se depășească numărul de bilete puse in vânzare pentru acel spectacol si va verifica sa nu mai fi fost bilete vândute pe același rând si număr.

Un casier va putea sa vadă lista cu toate biletele adăugate pentru un anumit spectacol.

Periodic, casierii pot exporta lista de bilete vândute pentru un anumit spectacol (in format csv sau xml). Se va implementa functionalitatea de export a biletelor (sau spectacolelor) in format JSON si in format CSV. Se va folosi design patternul Strategy pentru a implementa functionalitatea de export (Veti avea o interfata Exporter si doua clase care implementeaza aceasta interfata CsvExporter si JsonExporter). La runtime, utilizatorul poate alege formatul dorit pentru export. Folositi si Factory pattern pentru a crea obiectul specific cu care se instantiaza contextul din Strategy pattern.

- Datele vor fi salvate intr-o baza de date.

- Se va folosi pattern-ul arhitectural MVC la proiectarea si organizarea aplicatiei.

- Parolele vor fi salvate criptat in baza de date

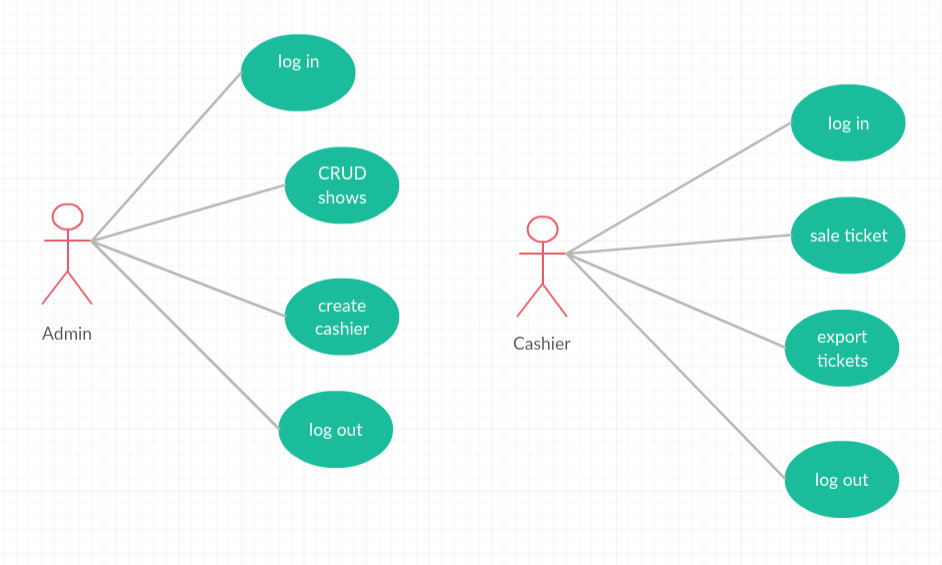
- Se va folosi o biblioteca ORM (ex: Hibernate pt Java)

## Non-functional Requirements

Disponibilitate: aceasta aplicatie desktop poate rula pe orice sistem modern ce dispune de un minim de resurse.

Extensibialitate: aplicatia poate fi extinsa cu alte functionalitati cum ar fi o descriere mai ampla a spectacolelor (inclusiv poze pentru spectacole), prezenta mai multor date pentru casieri (inclusiv poza de identificare) etc.

# Use-Case Model

**

*Use case: sale ticket*

*Primary actor: cashier*

*Main success scenario:*

* Introducere username si parola
* Validate username si parola
* Log in
* Se deschide fereastra in care se pot vedea tichetele vandute si spectacolele
* Se introduce id-ul spectacolului pentru care vindem bilet
* Se introduce numarul si randul locului specific biletului
* Se apara butonul “sale”

*Extensions:* daca validarea esuaza se arunca o exceptie

# System Architectural Design

## Architectural Pattern Description

Pentru realizarea aplicației am folosit arhitectura MVC (Model – View - Controller). Ideea este separarea interfeței utilizator într-un Model, un View (vedere, vizualizare) (creează afişajul, interacționând cu Modelul după nevoi), şi un Controller (răspunde la cererile utilizatorului, interacționând atât cu Vizualizarea cât şi cu Controlorul).

Arhitectură:

• **Model** -Această parte a controlatorului manipulează operațiunile logice și de utilizare a informației (trimisă dinainte de către rangul său superior), în cazul acestei probleme modelul este reprezentat de clasele care vor crea baza de date- Admin, Cashier, Show, Ticket, dar si clasele care fac operatii pe baza de date- AdminOperations, CashierOperations, ShowOp, TicketOp.

• **View**- Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controller. Interfața a fost realizată cu ajutorul bibliotecii Swing. Din View in cazul nostru fac parte clasele: Log, AdminView si CashierView, care definesc cele trei ferestre ale programului.

• **Controller** - Cu acest element putem controla accesul la aplicația noastră. În cazul aplicației de față avem 3 clase care controleaza aplicatia, cate una pentru fiecare window: ControllerInit (partea de log in), ControllerAdmin (controleaza fereastra specifica administratorului), ControllerCashier (controleaza fereastra folosita de cashier).

## Diagrams

Diagrama de pachete:

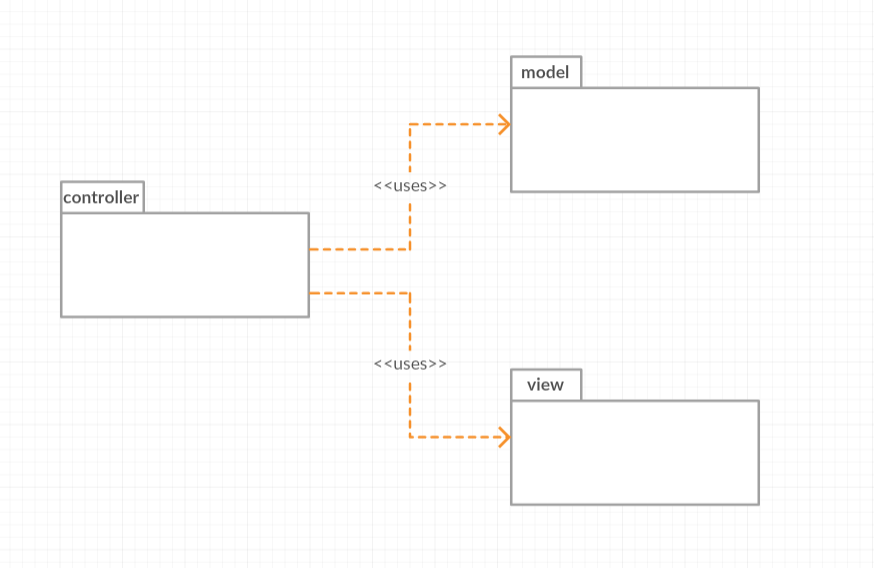
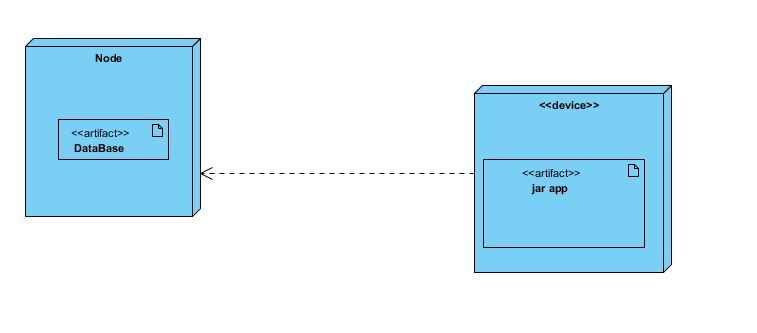
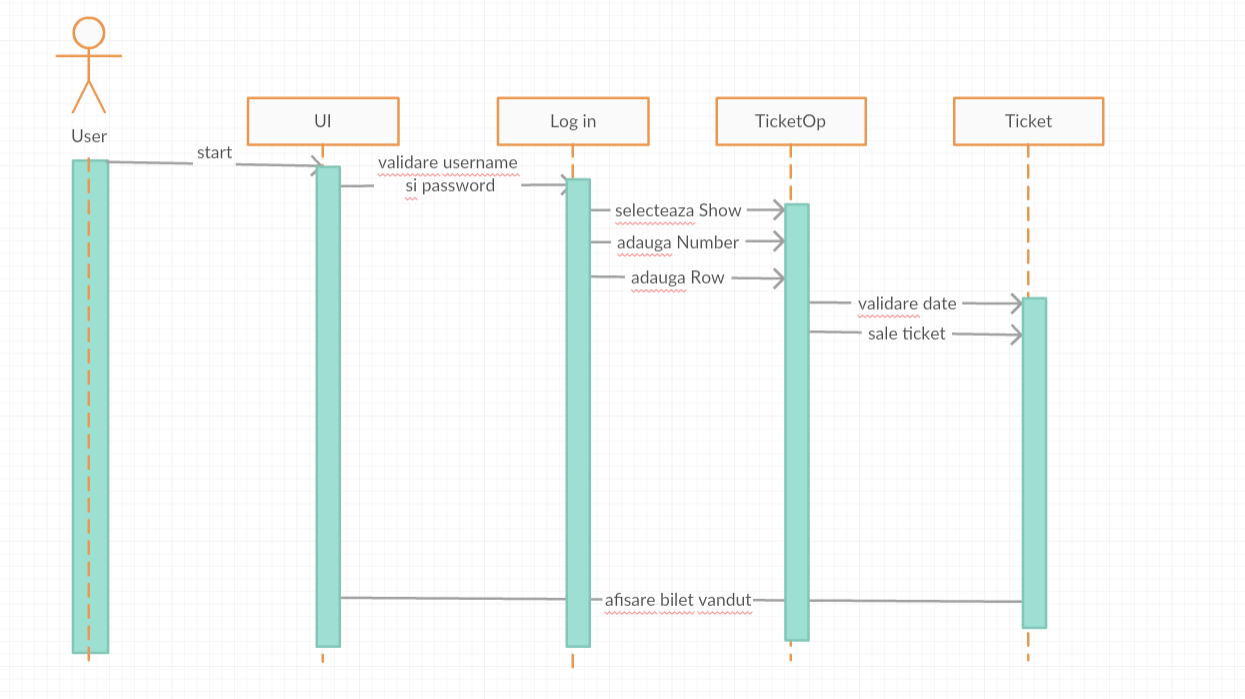
**

Diagrama de deployment:



# UML Sequence Diagrams

Diagrama de secventa pentru vinderea unui bilet la spectacol:



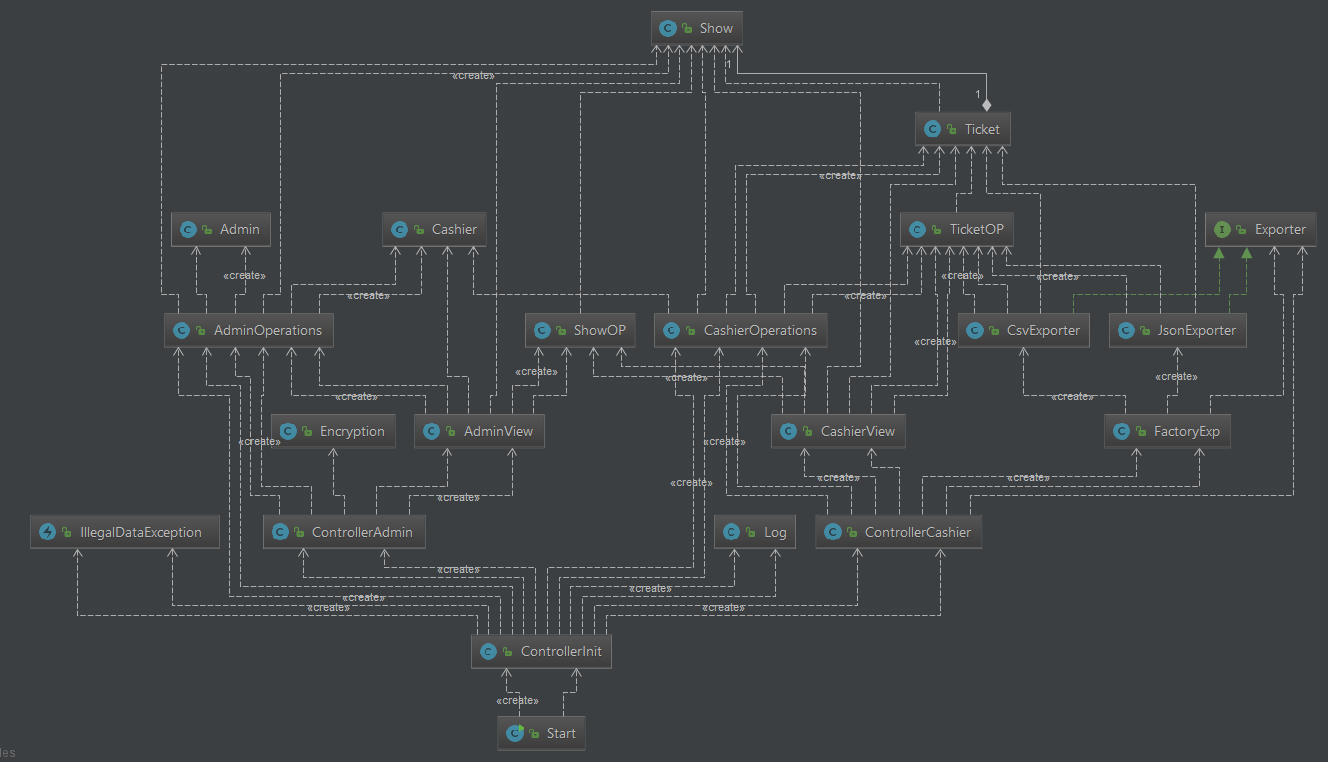
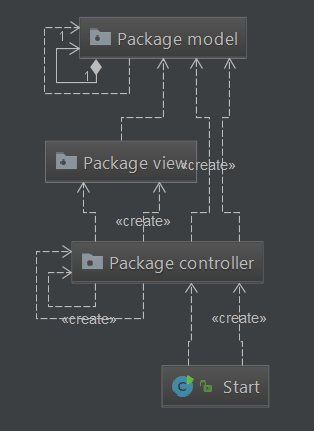
# Class Design

## Design Patterns Description

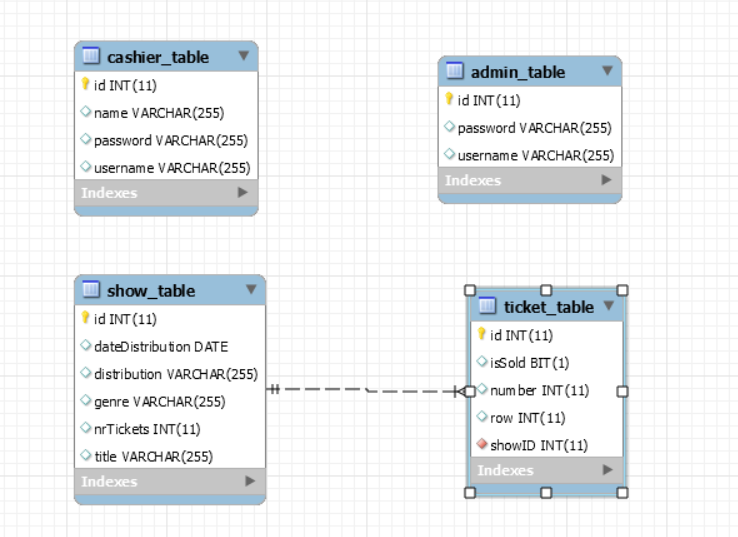
**Strategy** pattern: Acest pattern permite selectarea unui algoritm la executie. Este un model comportamental. In loc sa implementezi un singur algoritm direct, codul primeste instructiuni la rulare in legatura cu ce algoritm trebuie sa foloseasca. Acest pattern permite prin amanarea decizii algoritmului (pana la rulare) folosirea unui cod mai flexibil.

**Factory** pattern: este un model creator, deoarece acest pattern ofera una dintre cele mai bune modalitati de a crea un obiect. Prin acest pattern cream obiecte fara a expune logica clientului si ne referim la obiectele create (o interfata comuna)) folosind o metoda getExporter().

## UML Class Diagram

**

# Data Model

**

# Bibliography

* <https://dzone.com/articles/java-the-strategy-pattern>
* <https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm>
* <https://www.tutorialspoint.com/design_pattern/factory_pattern.htm>
* <https://www.roseindia.net/hibernate/hibernate4/spring_hibernate_encrypted_password.shtml>