Aplicatie pentru gestionarea spectacolelor Operei Nationale din Cluj

Aplicatie realizata de

Bursuc Andrei

Grupa 30234

Contents

[1. Requirements Analysis 3](#_Toc512450164)

[1.1 Assignment Specification 3](#_Toc512450165)

[1.2 Functional requirements 3](#_Toc512450166)

[1.3 Non-functional Requirements 3](#_Toc512450167)

[2. Use-Case 4](#_Toc512450168)

[3. System Architectual Design 5](#_Toc512450169)

[3.1 Architectural Pattern Description 5](#_Toc512450170)

[3.2 Diagram 6](#_Toc512450171)

[4. UML Sequence Diagrams 6](#_Toc512450172)

# Requirements Analysis

## Assignment Specification

Acesta tema urmareste realizarea unei aplicatii ce ajuta la gestionarea spectacolelor si a biletelor care se pun in vanzare pentru spectacole. Aplicatia trebuie sa aiba doua tipuri de utilizatori: casierii, ce se pot loga in sistem, adauga bilete pentru un anumit spectacol, iar periodic pot exporta lista de bilete vandute pentru un anumit spectacol; administratorii ce vor gestiona lista de spectacole si vor putea sa creeze conturi pentru casieri.

## Functional requirements

Datele vor fi stocate intr-o baza de date. Foloseste modelul de design Layers pentru organizarea aplicatiei. 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 pentru Java). 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. 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.

## 1.3 Non-functional Requirements

Disponibilitate: aceasta apicatie desktop poate rula pe orice sistem modern ce dispune de un minim de resurse, ea neavand nevoie de o conexiune la internet

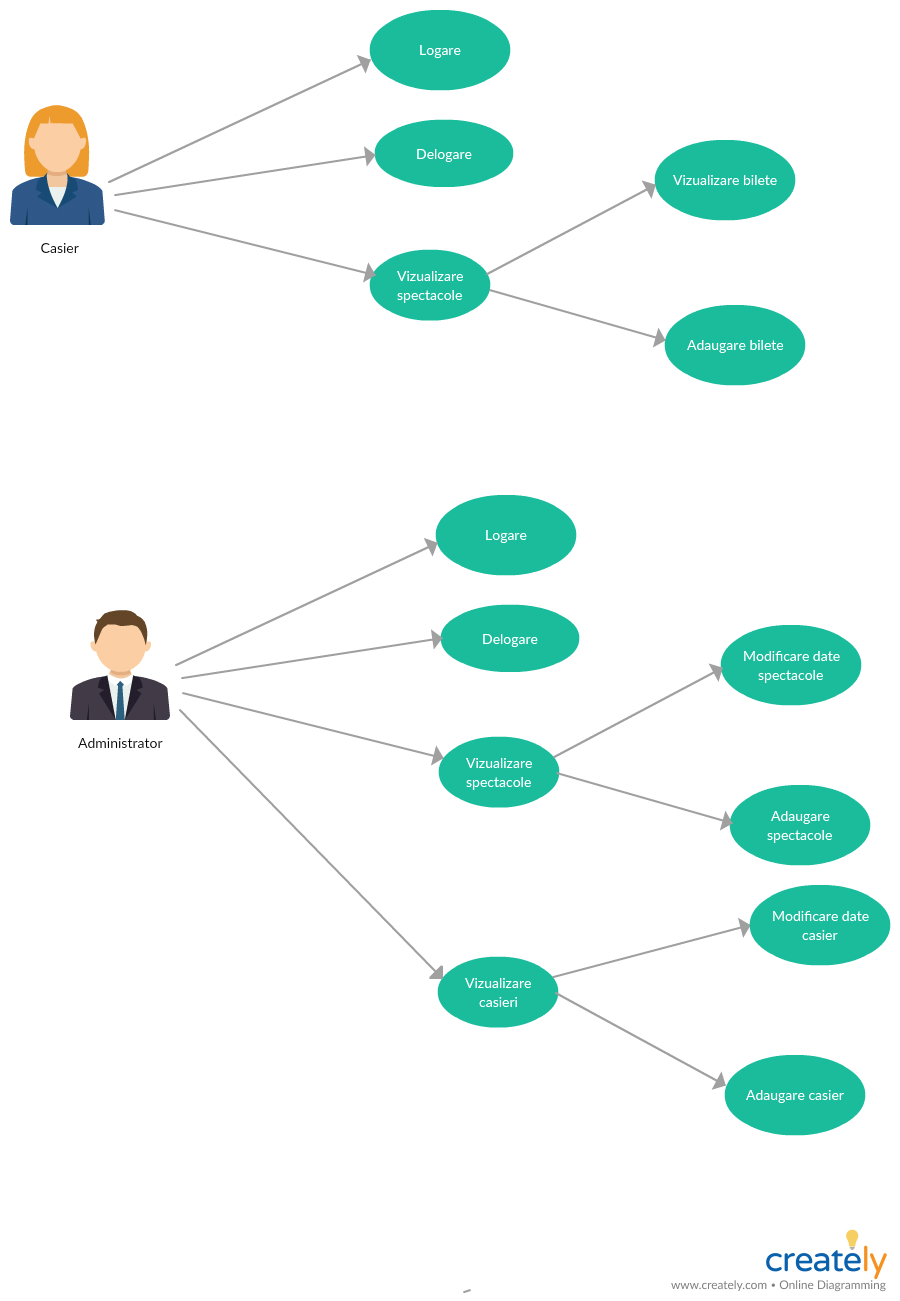
Extensibialitate: aplicatia poate fi extinsa cu alte functionalitati cum ar fi o descriere mai ampla a spectacolelor (inclusiv poze pentru spectacole), prezenta mai multor date pentru casieri (inclusiv poza de identificare) etc.

Mentenanta: imbunatatirea aplicatiei se poate face relativ usor, codul fiind bine structurat, la o imbunatatire majora nefiind nevoie sa fie modificat tot codul, ci doar din anumite layere.

Portabilitate: aplicatia ruleaza pe orice sistem de operare atata timp cat exista un mediu Java Runtime Environment instalat. Se recomanda utilizarea celei mai noi verisuni de JRE.

Securitate: Conexiunea la baze de date este una securizata nefiind accesul la ea de catre utilizatorii neautorizati, iar parola este criptata in baza de date cu metoda SHA256.

# Use-Case



# System Architectual Design

## Architectural Pattern Description

Pentru realizarea aplicației am optat pentru utilizarea arhitecturii MVC (Model ). 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 după nevoi.

Arhitectură:

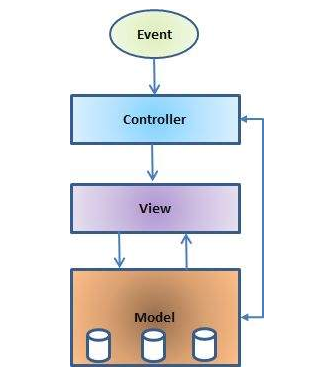
• **Model** -Această parte a controlatorului manipulează operațiunile logice și de utilizare a informației (trimisă dinainte de către rangul său superior), în cazul acestei probleme modelul este reprezentat de: clasa Person ce defineste clientii, clasa abstracta Account ce defineste conturile clientilor, clasele ce extind Account, SpendingAccount si SavingAccount ( ele definesc fiecare in parte tipurile conturilor), interfata BankProc si clasa Bank ce o implementeaza (ea defineste banca cu clientii si conturile lor) si interfata IEL cu clasa EventListener ce monitorizeaza actiunile din program si notifica clientii.

• **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 Javax.Swing utilizând un plugin special pentru IntelIJ denumit JFormDesigner, ce permite realizarea mai ușoară a unei interfețe atractive pentru utilizator și ușor de folosit. Din View in cazul nostru fac parte clasele: ClientView, BankView si StartingPanel care definesc cele trei ferestre ale programului.

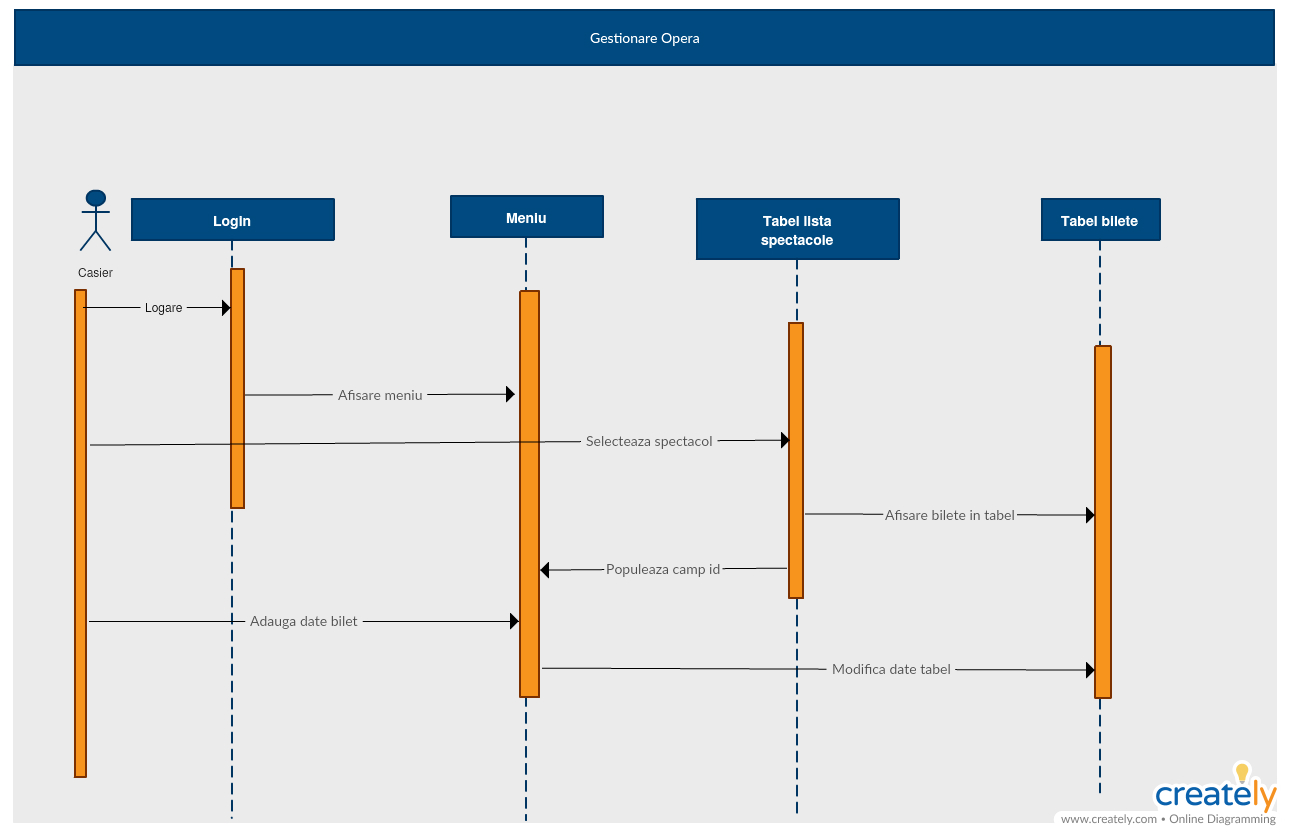
• **Controller** - Cu acest element putem controla accesul la aplicația noastră. Pot fi fișiere, scripts sau programe, in general orice tip de informație permisă de interfață. În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp. În cazul aplicației de față constructorul controller-ului primeste ca parametrii trei obeicte de tipul: Bank, StartingPanel si listener si are rolul de a lega intre ele ferestrele din program si de a permite listener-ului sa fie transmis la celelalte obiecte din controller.

Codul a fost scris si compilat utilizând IDE-ul IntelIJ IDEA, mediu de dezvoltare software dezvoltat de catre compania JetBrains ( cunoscută anterior sub numele de IntelIJ ). El vine in doua variante ”Community Edition” și ”Ultimate Edition”, aceasta din urmă fiind disponibilă gratis studenților. Prima versiune a IntelIJ a fost lansată in Ianuarie 2001 și a fost unul dintre primele IDE-uri pentru java care dispunea de funcții avansate de ” Code Navigation” și ”Code Refactoring”. Ultima versiune 12.1 dispune de suport pentru Java 8, posibilitatea de dezvoltare de interfață grafică pentru Android, Play 2.0 și Scala.

## Diagram



# UML Sequence Diagrams



# Class Design

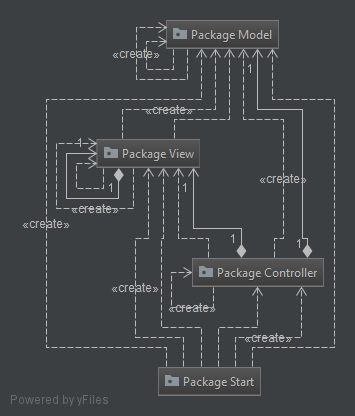
## Design Patterns Description

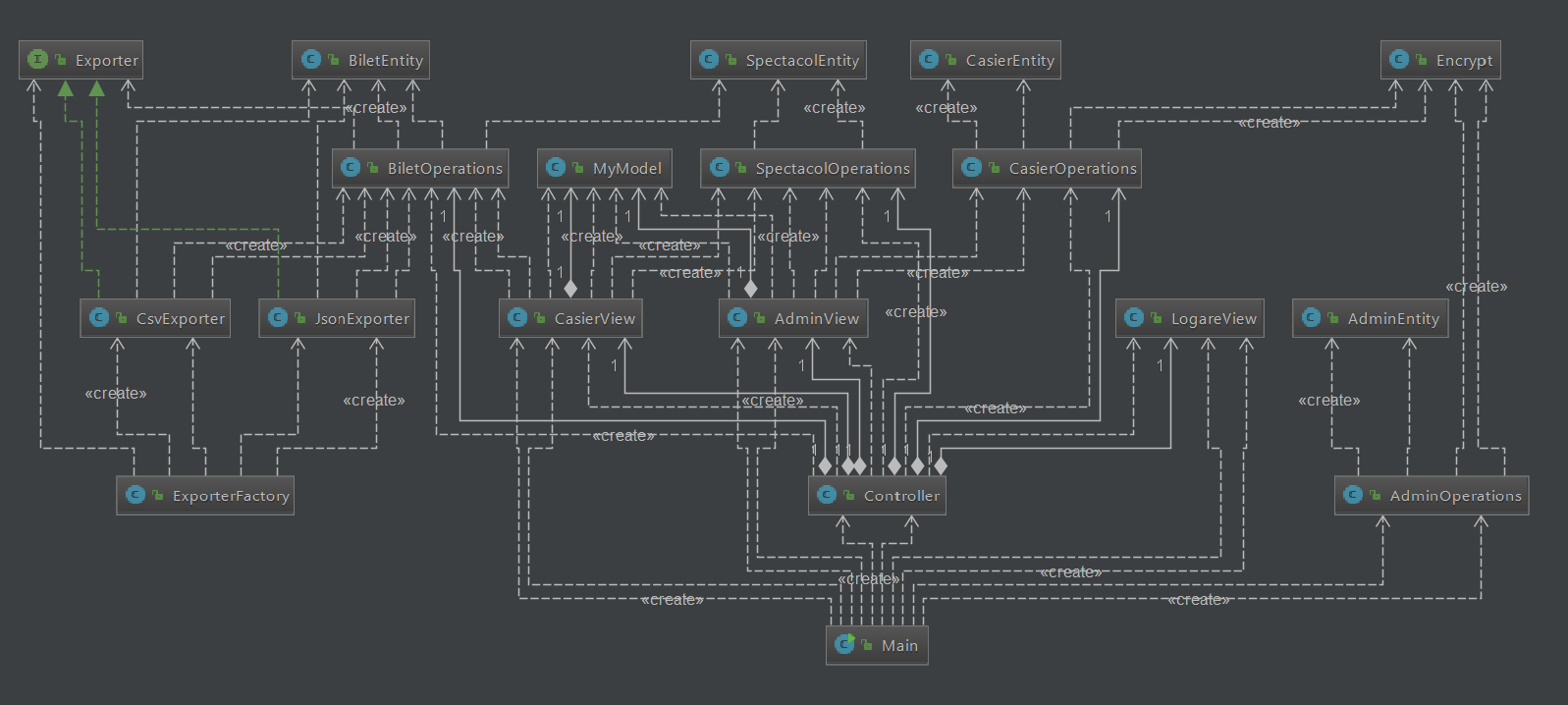
Strategy pattern: Acest pattern permite selectarea unui algoritm la rulare. In loc sa implementezi un singur algoritm direct, codul primeste instructiuni la rulare in legatura cu ce algoritm trebuie sa foloseasca. Acest pattern permite prin amanarea decizii algoritmului (pana la rulare) folosirea unui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

Factory pattern: Prin acest pattern cream obiecte fara a expune logica clientului si ne referim la obiectele create folosind o metoda factory.

## 5.2 UML class diagram

In continuare este prezentata diagrama UML a pachetelor folosite in program iar mai apoi diagrama UML cu toate clasele

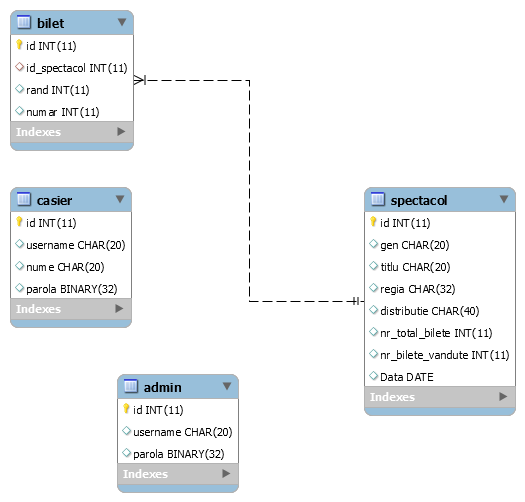




# Data Model

Baza de date este alcatuita din 4 tabele: Admin, Casier, Spectacol si Bilet. Tabelul admin este definit de campurile: id, username, parola, logat .Tabelul casier este definit de campurile: id, username, nume, parola, acest tabel ne ajuta sa stocam datele despre casier. Tabelul Spectacol cu campurile: id, gen, titlu, regia, distributie, nr\_total\_bilete, nr\_bilete\_vandute, Data, pret stocheaza datele despre spectacole; Tabelul bilet definit de: id, id\_spectacol, rand, numar; acest tabel este legat de tabelul spectacol prin cheia straina id\_spectacol si in el sunt salvate biletele pentru fiecare spectacol in parte.

## Database diagram



# Bibliography

1. <http://docs.oracle.com/javase/>
2. <http://stackoverflow.com/>
3. Kathy Sierra, Bert Bates. SCJP Sun Java Certified Programmer for Java 6. McGraw Hill, USA 2008
4. https://dzone.com/articles/java-the-strategy-pattern