National Theater

Student: Cornescu Andreea Mara

**Group: 30431**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

Design and implement an application for the National Theater of Cluj. The application has two types of users: cashier and administrator, which must provide a username and a password to connect to the application.

# Functional Requirements

The administrator user is able to perform the following operations:

* CRUD on cashiers’ information;
* CRUD of the list of shows that are performed at the theater;
* Export all the tickets that were sold for a certain show to a csv or xml file;

The cashier can:

* Sell tickets to a show;
* See all the tickets for a show;
* Cancel a reservation;
* Edit a seat;

The system notifies if the number of tickets per show has been exceeded;

# Non-functional Requirements

* The data will be stored in a database.
* Using **factory method**, export the data to csv or xml;

Security requirements:

* Passwords are encrypted when stored in the database by using a one way encryption algorithm;
* Ability to configure a user’s rights to enable them only to access the functions that they have a right to;
* Ensure only valid data is stored in the database;

Usability:

* Easy-to-use application;

2. Use-Case Model

Use case: Sell ticket

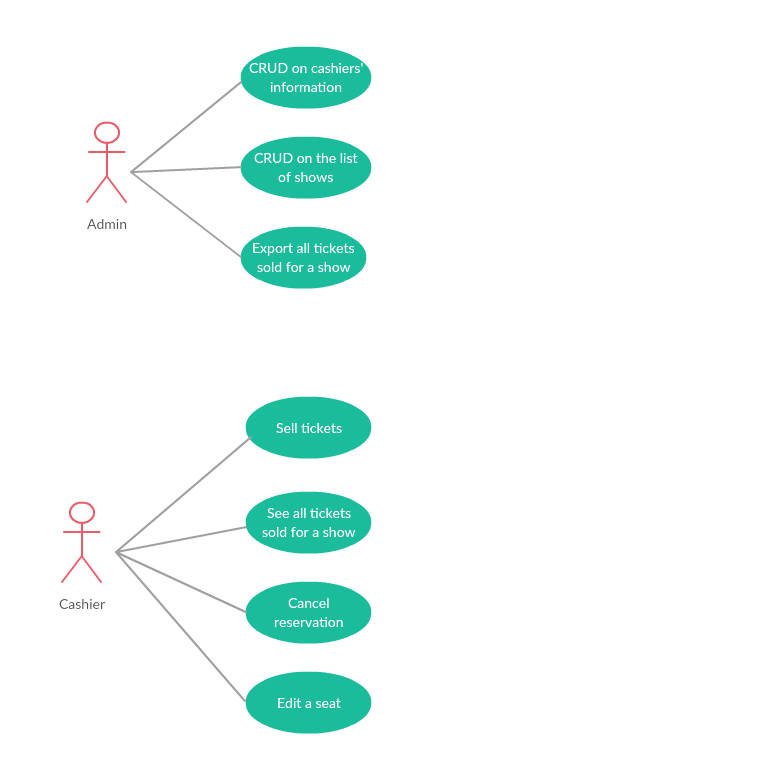
Level: User-goal level

Primary actor: Cashier

Main success scenario: Select show, introduce ticket information (seat, price), sell ticket

Extensions: No tickets available for the show => fail selling

Use-case diagram:



3. System Architectural Design

**3.1 Architectural Pattern Description**

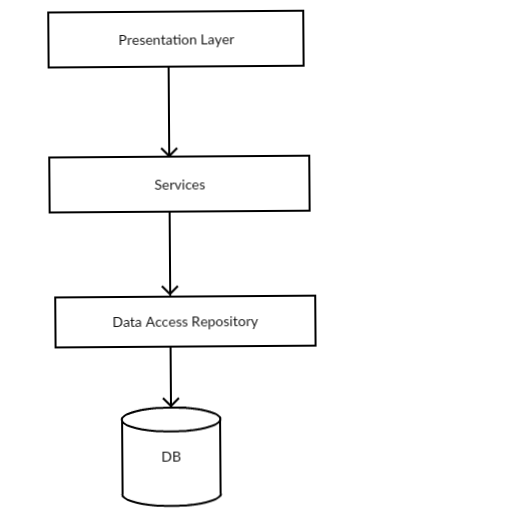
The application is designed following the Layered Architecture pattern. Components within the layered architecture are organized into horizontal layers, each layer having a well defined role within the application. Each layer provides a set of services which can be used by the layer above.

The architecture pattern is composed of the following layers:

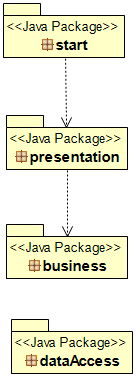
* Presentation Layer – responsible with the UI;
* Services Layer – business logic behind the application;
* Data Access Repository – provides access to the database and performs specific operations;

**3.2 Diagrams**

**Conceptual architecture of the system**

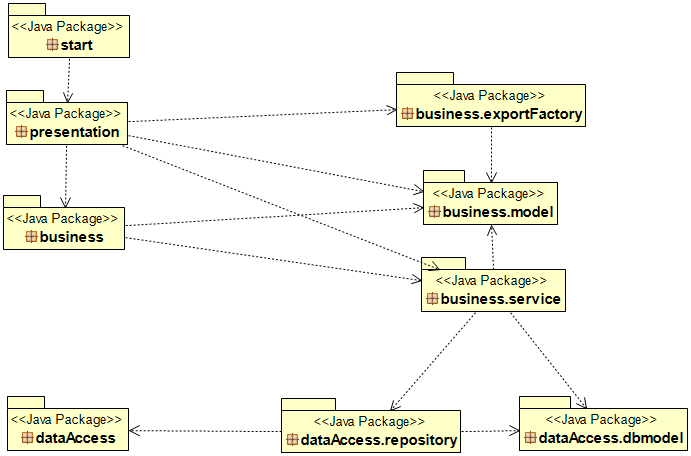
****

**Package Diagram**

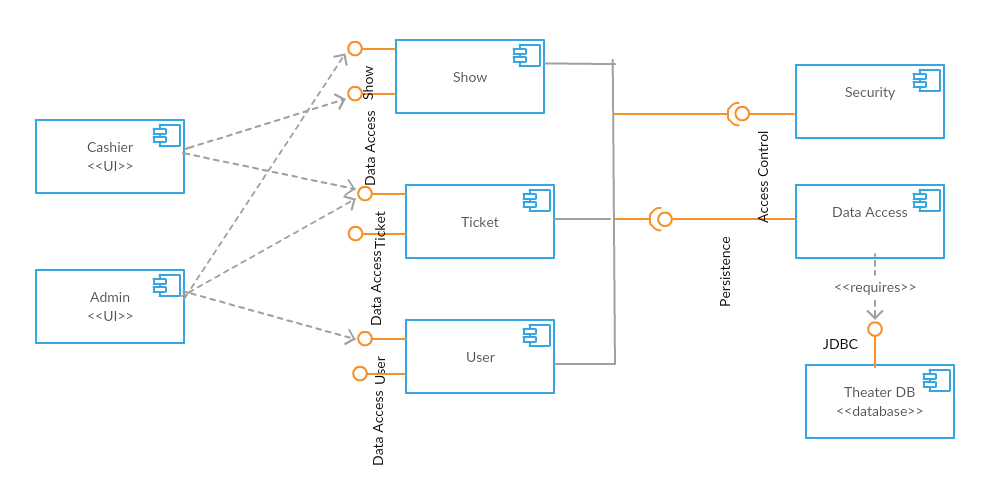
****

Presentation Package contains: export factory package, model package and service package.

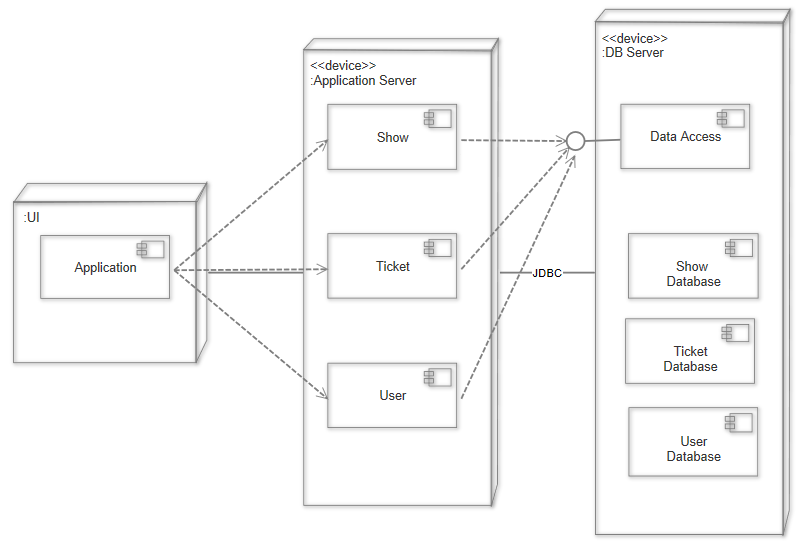
Data Access Package has the following sub-packages: repository and dbmodel.

****

**Component Diagram**

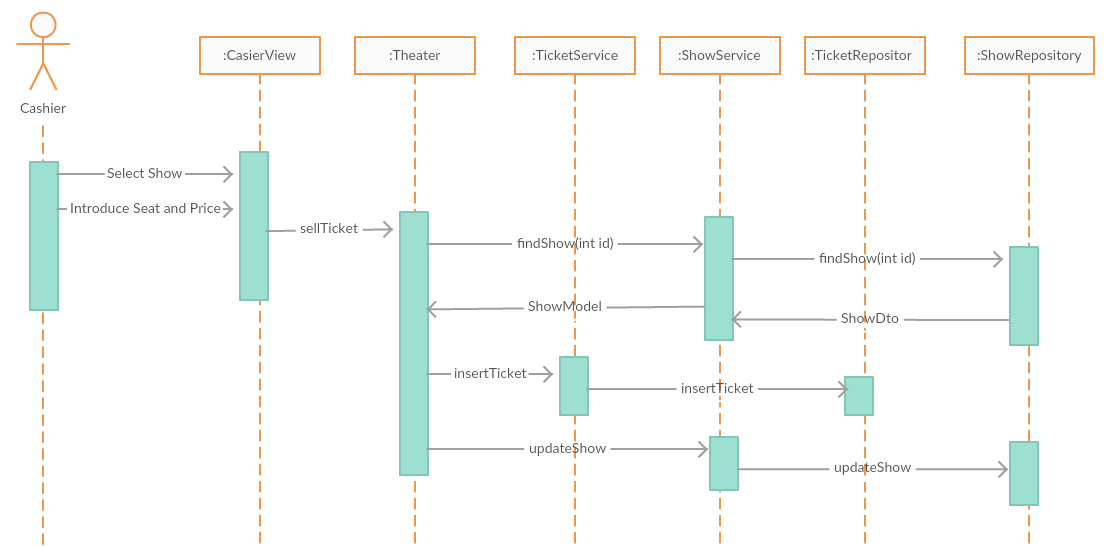
****

**Deployment diagram**

****

4. UML Sequence Diagrams

Sequence diagram for selling a ticket:



5. Class Design

**5.1 Design Patterns Description**

**Factory Method Design Pattern**

Factory method is used to create objects without exposing the creation logic to the client. An interface or an abstract class is used to create an object, but the choice of what class to instantiate is left to the subclasses.

Factory method is used to export the number of tickets sold for a show.

**Singleton Pattern**

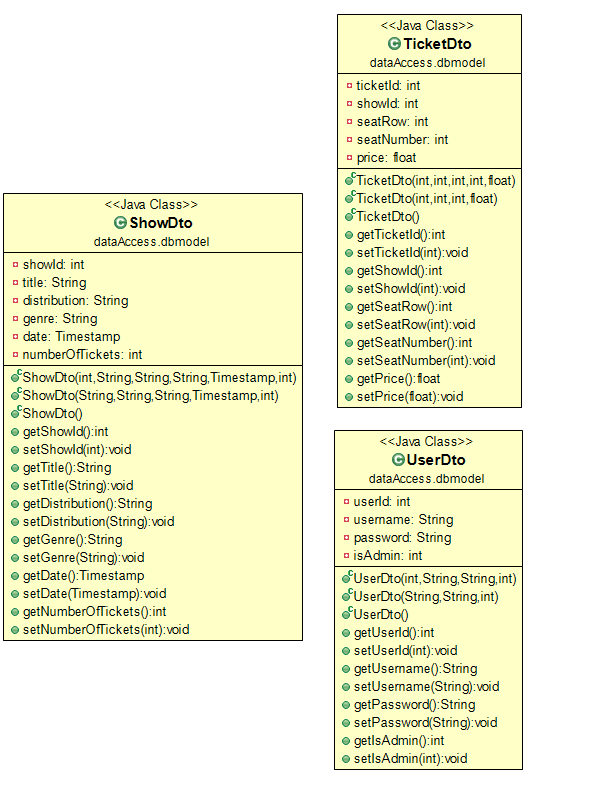
Singleton pattern is used to ensure that only a single object of a class can be created. The class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class. Singleton class hides its only instance method behind a static method which returns a reference to the single instance.

Singleton Pattern is used to create the database connection.

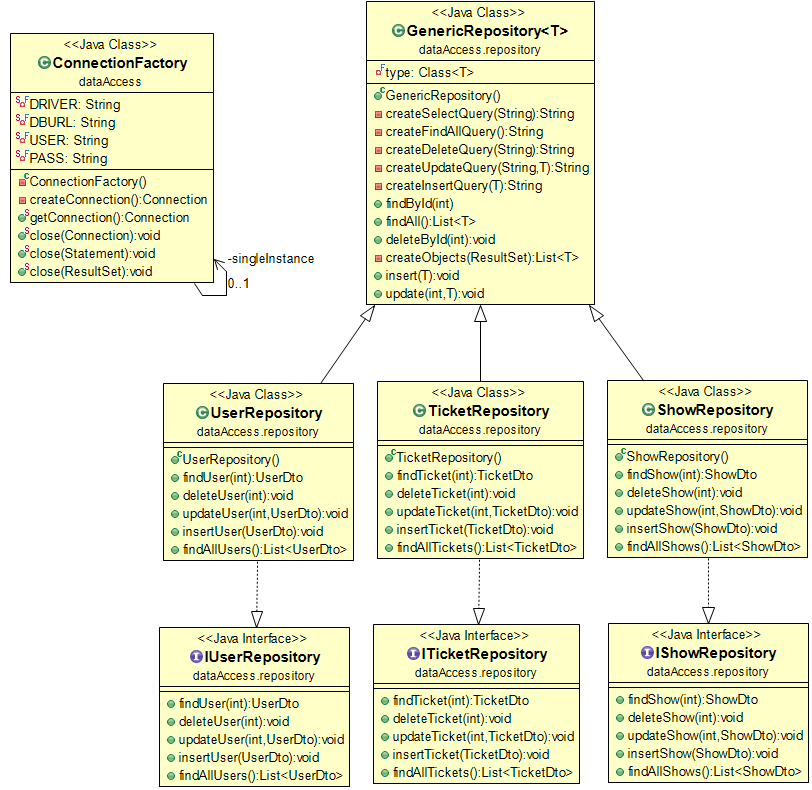
**5.2 UML Class Diagram**

**Data access package** contains the following packages:

* dbmodel



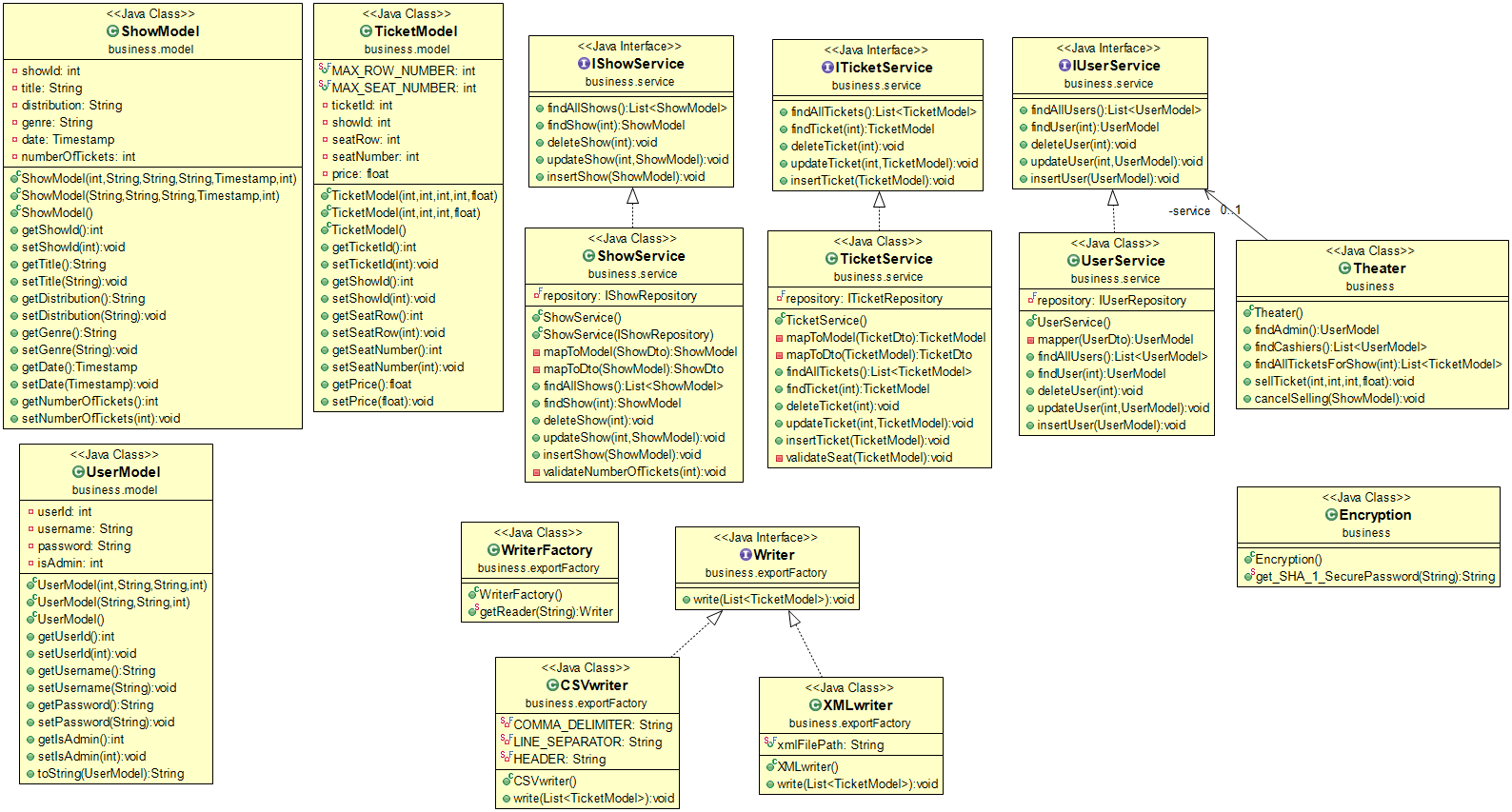
* repository



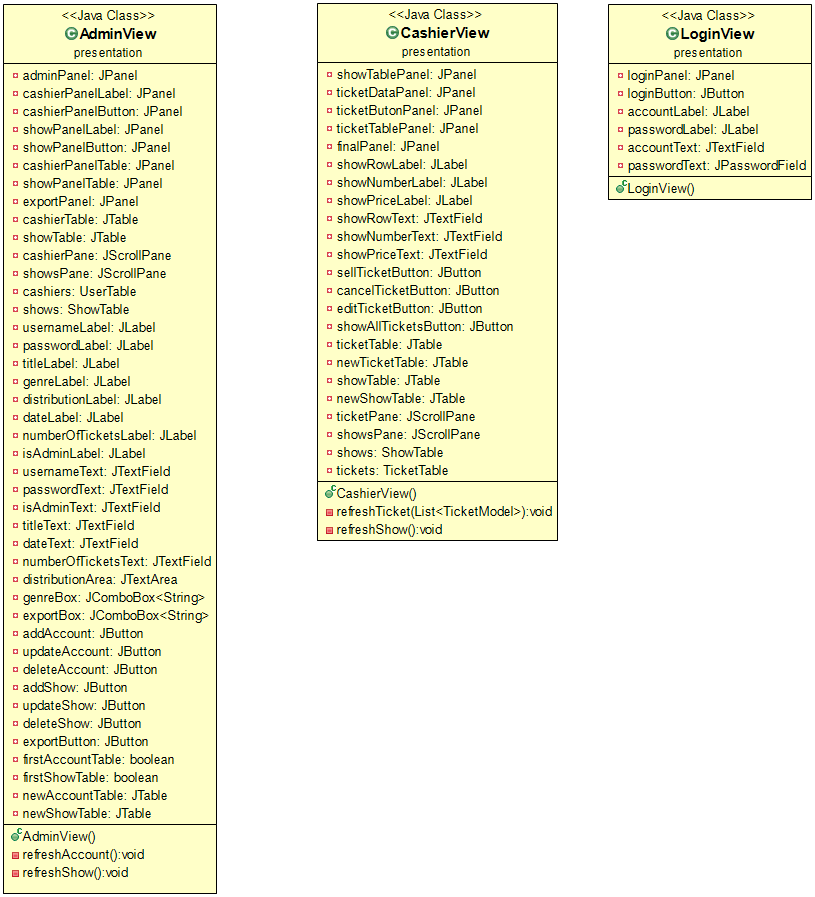
**Business package** contains the following packages:

* export factory – factory method used to export the tickets sold for a show in two different formats;
* model;
* service;

Also, it contains the Theater class which performs operations like: selling a ticket, finding if the user is admin or cashier, and the Encryption class.

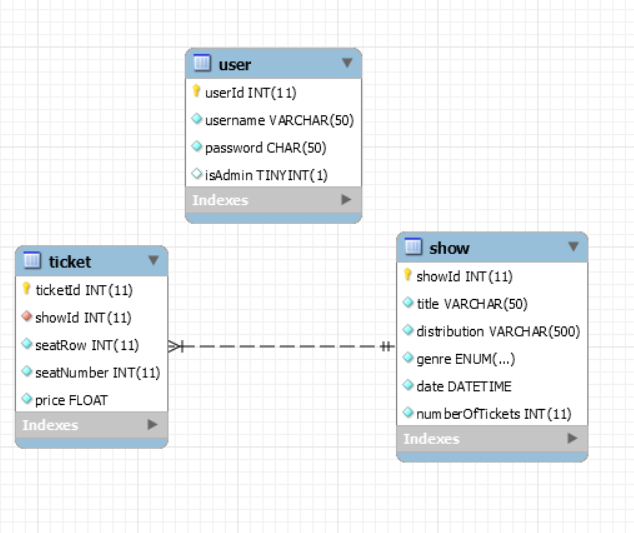


**Presentation package** contains the view for login, admin, and cashier.



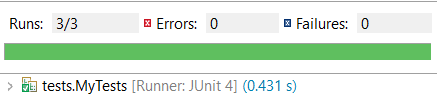
6. Data Model

Three models were used in this application: user, show, and ticket.



7. System Testing

Unit testes were designed to verify if the methods for selling a ticket, password encryption, and find all shows execute correctly. A ticket cannot be sold if the selected place has already been given to someone else, or if there are no tickets available for that shows. To find all show a mocking method for the database was used.



8. Bibliography

* Factory pattern: <https://www.tutorialspoint.com/design_pattern/factory_pattern.htm>
* Singleton pattern: <https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm>
* Component diagram: <http://www.agilemodeling.com/artifacts/componentDiagram.htm>
* Deployment diagram: <http://www.agilemodeling.com/artifacts/deploymentDiagram.htm>
* Layers architecture: <https://towardsdatascience.com/software-architecture-patterns-98043af8028>
* Mocking testes: <https://www.mkyong.com/unittest/unit-test-what-is-mocking-and-why/>