Student: Adela Mera

**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 should have two types of users (a cashier user represented and an administrator) which must provide a username and a password to use the application.

# Functional Requirements

* CRUD on cashiers’ information
* CRUD on the list of shows that are performed at the theater
* export all the tickets that were sold for a certain show
* sell tickets to a show
* cancel a reservation
* edit the seat
* see all the tickets that were sold for a show

# Non-functional Requirements

## Performance

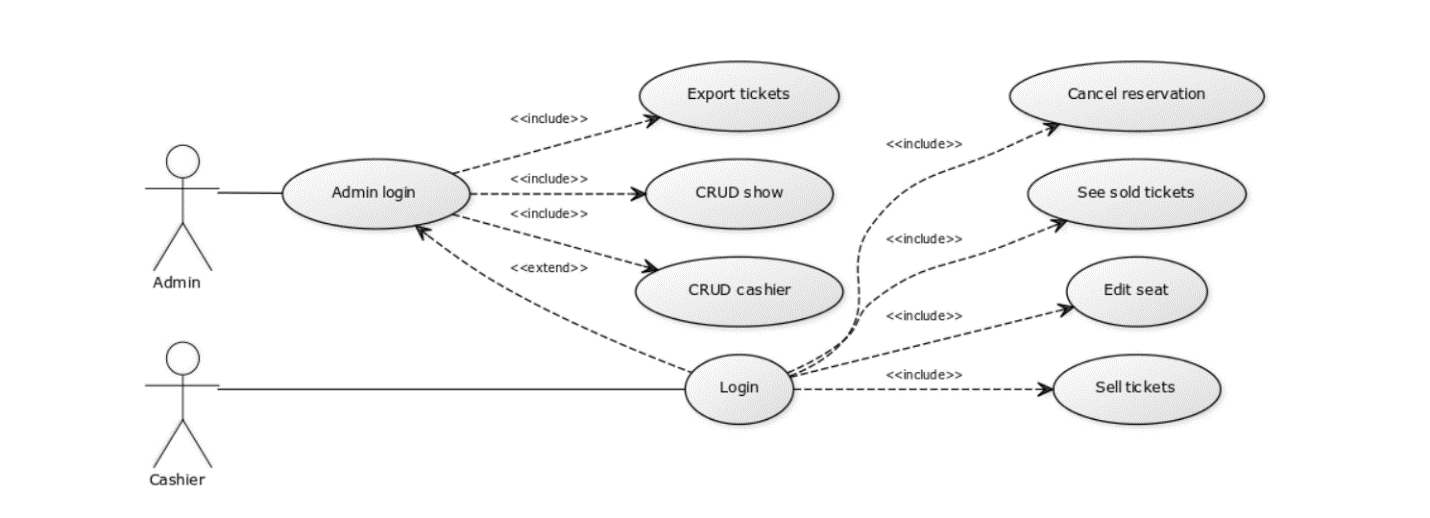
* The operations should take the least time possible to complete
* Minimal time between click and system response
* Production of a simple export shall take less than 5 seconds for 95% of the cases.

**Availability**

* The system shall meet or exceed 99.99% uptime.

## Security

* Password will never be displayed
* Password will be encrypted
* Cashier’s information will only be accessible to administrators

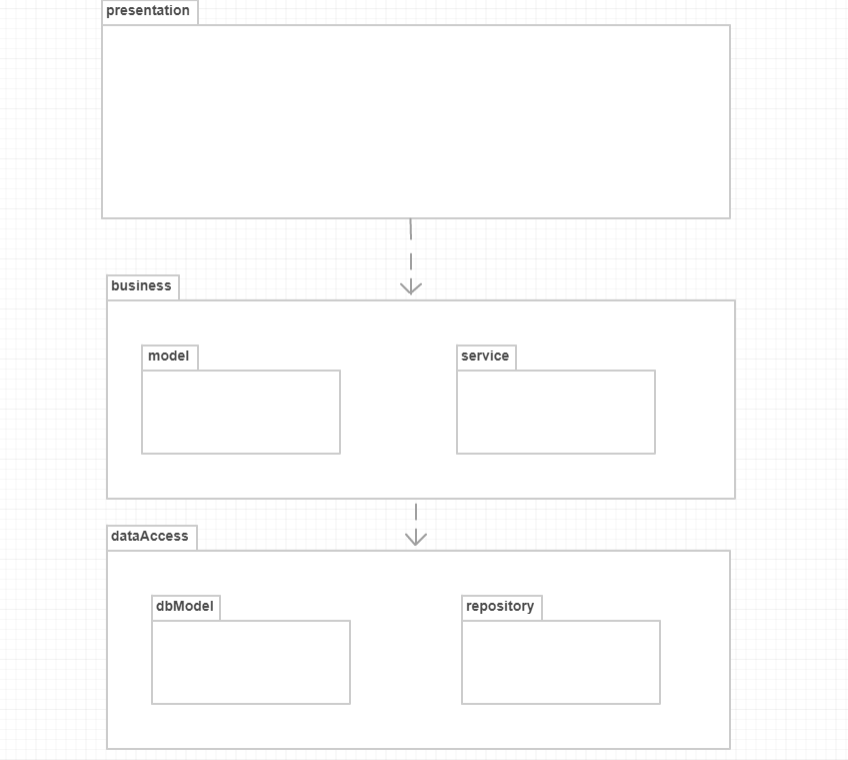
**2. Use-Case Model

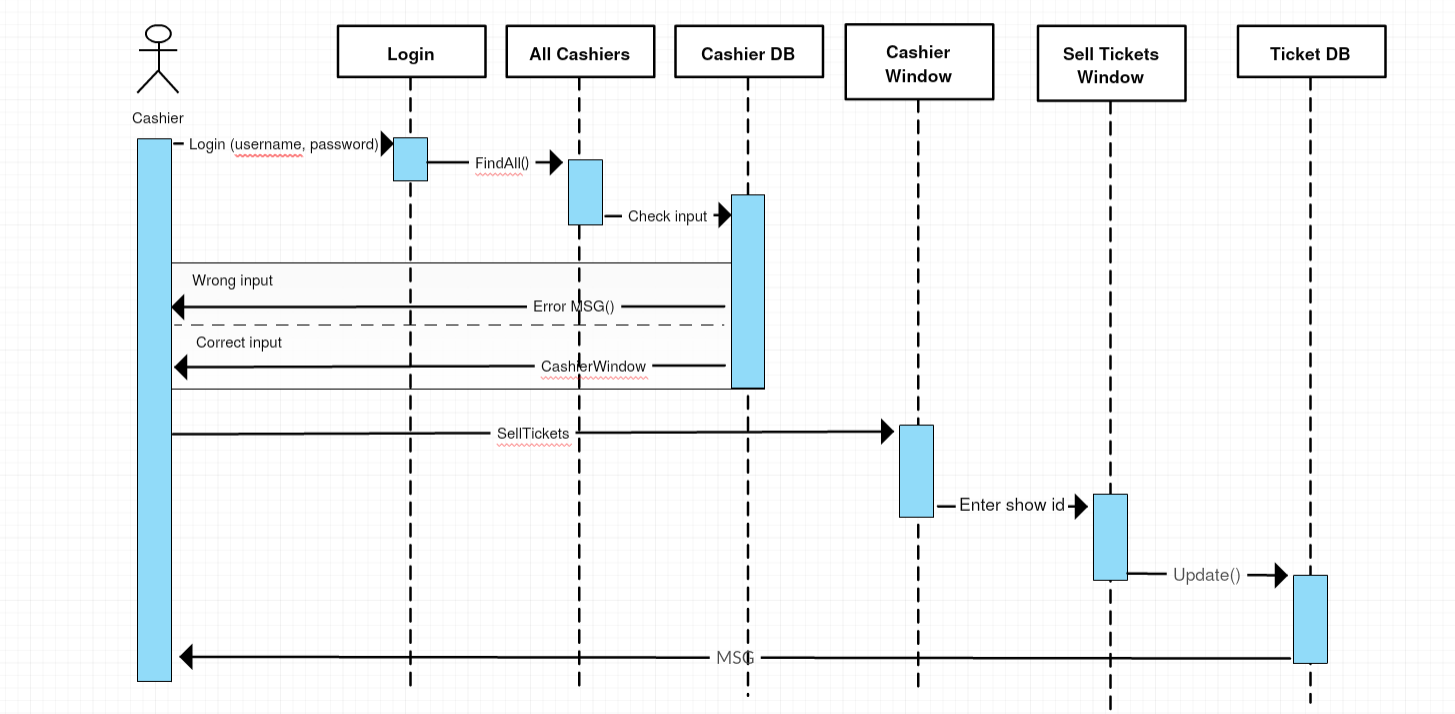
3. System Architectural Design

**3.1 Architectural Pattern Description**

A layered architecture pattern was used in order to obtain reusability which is essential within the OOP paradigm. In this pattern, each layer has a specific role and responsibility within the application. For example, the presentation layer is responsible for handling all user interface logic, whereas the business layer is responsible for executing specific business rules associated with the request. As a request moves from layer to layer, it must go through the layer right below it to get to the next layer below that one. So, all the components which are responsible for data visualization will be placed in the presentation layer, all business logic rules will be encapsulated within the business layer and the data access layer is responsible for database access.

**3.2 Diagrams**

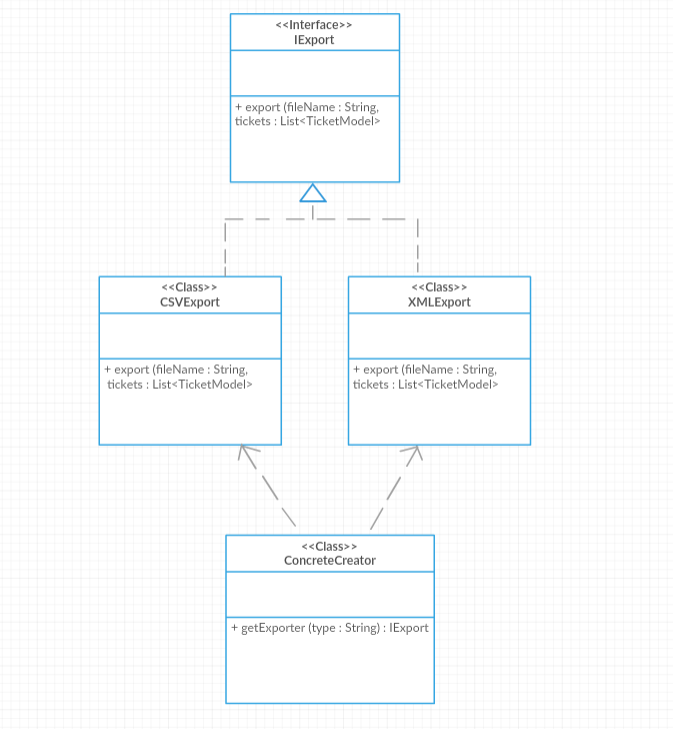
**

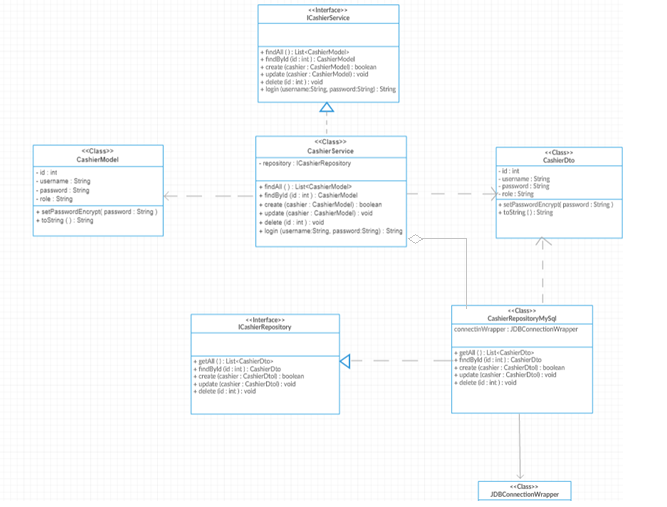
4. UML Sequence Diagrams

5. Class Design

**5.1 Design Patterns Description**

The Factory Method design pattern was used to export ticket data in a CSV or XML file. The Factory Method is a creational design pattern that suggests replacing direct object creation with a call to a special "factory" method, in which the constructor call is moved.

**5.2 UML Class Diagram**



6. Data Model

In order to transfer data from the database to the user interface Data Transfer Objects were used. The difference between data transfer objects and [business objects](https://en.wikipedia.org/wiki/Business_object) is that a DTO does not have any behavior except [getters](https://en.wikipedia.org/wiki/Mutator_method) and setters. In other words, DTOs are simple objects that should not contain any business.

7. System Testing

In order to avoid having a slow unit test because we have to start a database in order to get data and to avoid having a unit test that is not isolated because it always depends on a database, we used mocking. Mocking is creating objects that mimic the behavior of real objects. In a [unit test](https://en.wikipedia.org/wiki/Unit_test), mock objects can [simulate](https://en.wikipedia.org/wiki/Simulation) the behavior of complex, real objects and are therefore useful when a real object is impractical or impossible to incorporate into a unit test.

8. Bibliography

<https://en.wikipedia.org/wiki/Mock_object>

<https://www.mkyong.com/unittest/unit-test-what-is-mocking-and-why/>

<https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

<http://www.dofactory.com/net/design-patterns>