Assignment1

Student: Chindriș Mihai

**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

Use JAVA/C# API to 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.

# 1.2 A cashier can see all the tickets that were sold for a show, cancel a reservation or edit the seat

# Functional Requirements

The administrator user can perform the following operations:

1CRUD on cashiers’ information.

1. CRUD on the list of shows that are performed at the theater. Keep track of the Genre (Opera, Ballet), Title, Distribution list (a long string is enough), Date of the show and the Number of tickets per show.
2. From time to time he can export all the tickets that were sold for a certain show (e ither in a csv or xml file).

The cashier can perform the following operations:

1. Sell tickets to a show. A ticket should hold information about the seat row and seat number.
2. The system should notify the cashier that the number of tickets per show was not exceeded.

# Non-functional Requirements

1The data will be stored in a database.

1. Use the Layers architectural pattern to organize your application.
2. Passwords are encrypted when stored to the database with a one-way encryption algorithm.
3. Provide unit tests for the number of tickets for show exceeded validation and the encryption algorithm.
4. Use **factory method (not factory)** for export to csv/xml.

2. Use-Case Model

*Use case: sell ticket*

*Level: user-goal level*

*Primary actor: cashier*

*Main success scenario: Ticket sold to a show*

*Extensions: show is fully booked*

*Use case: update ticket*

*Level: user-goal level*

*Primary actor: cashier*

*Main success scenario: Ticket row or seat updated*

*Extensions: seat is already taken*

*Use case: delete ticket*

*Level: user-goal level*

*Primary actor: cashier*

*Main success scenario: Ticket deleted*

*Extensions: --*

*Use case: crud on shows*

*Level: user-goal level*

*Primary actor: admin*

*Main success scenario: operation successful*

*Extensions: something happens*

*Use case: crud on users*

*Level: user-goal level*

*Primary actor: admin*

*Main success scenario: operation successful*

*Extensions: something happens*

*Use case: export tickets to show*

*Level: user-goal level*

*Primary actor: admin*

*Main success scenario: tickets exported*

*Extensions: --*

*Use case: encrypt password*

*Level: function*

*Primary actor: server*

*Main success scenario: password hashed*

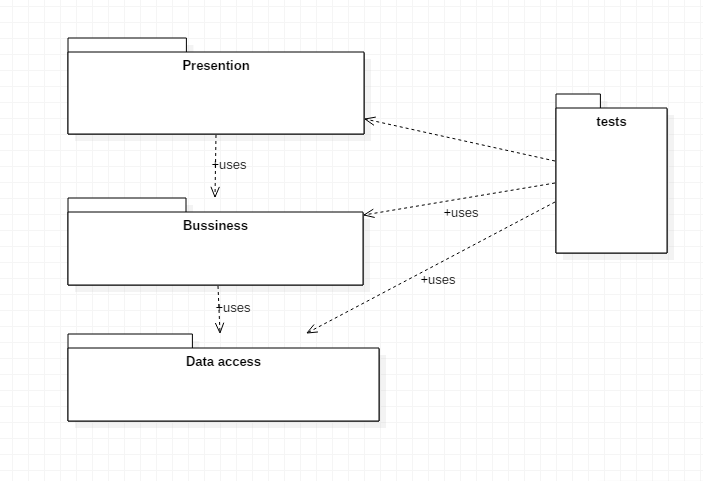
*Extensions: password is empty*

3. System Architectural Design

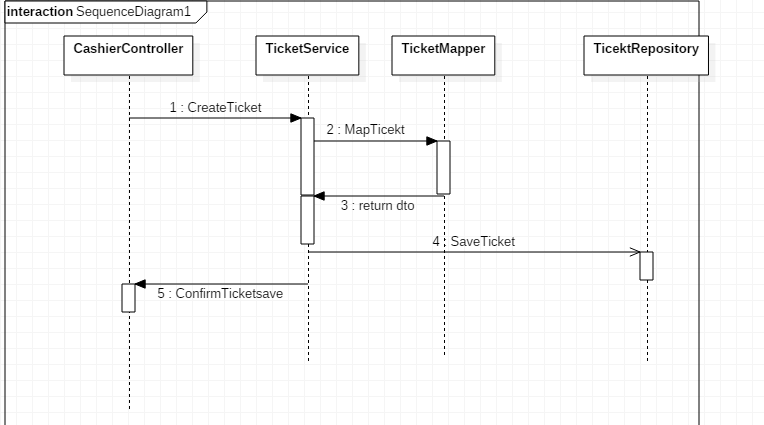
**3.1 Architectural Pattern Description**

*For this assignment the layers pattern was used, which is grouping functionalities on layers. The top level layers depend on lower level layers.*

**3.2 Diagrams**



4. UML Sequence Diagrams

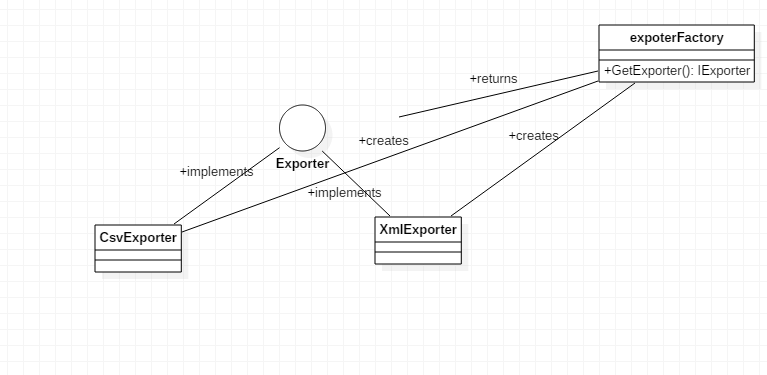


5. Class Design

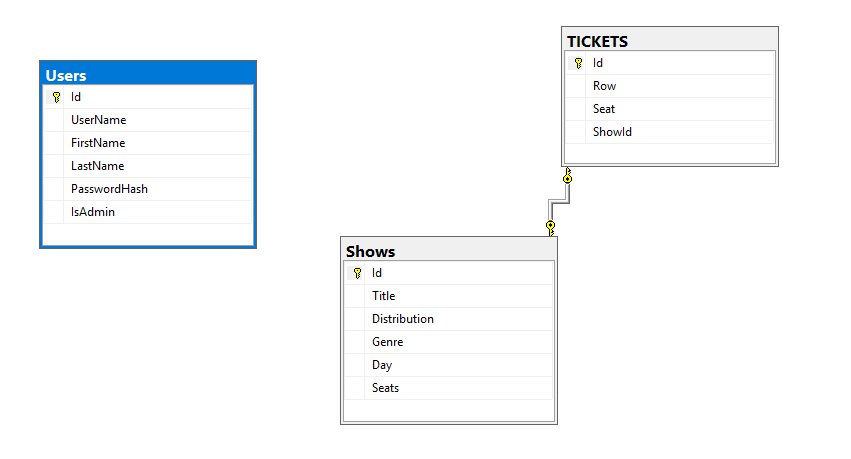
**5.1 Design Patterns Description**

*For this assignment the factory method was used, which creates a new object based on some parameter*

**5.2 UML Class Diagram**



6. Data Model



7. System Testing

Unit tests were created for verifying the hashing algorithm(check that the result is always 64 characters long) and ticket service(check whether more tickets were sold than available)

8. Bibliography