<Assignment 1>

Analysis and Design Document

Student: Terec Razvan Dan

**Group: 30231**

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 C# Web.API/ JAVA SWING (optional) 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.

# Functional Requirements

* The administrator user can perform the following operations:
* CRUD on cashiers’ information.
* 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.
* He can export all the tickets that were sold for a certain show (in a csv file).
* The cashier can perform the following operations:
* Sell tickets to a show.
* The system notify the cashier that the number of tickets per show was not exceeded when trying to book a new seat.
* A cashier can see all the tickets that were sold for a show, cancel a reservation or edit the reserved seat.

# Non-functional Requirements

2. Use-Case Model

***Use-Case description format:***

*Use case: Sell ticket*

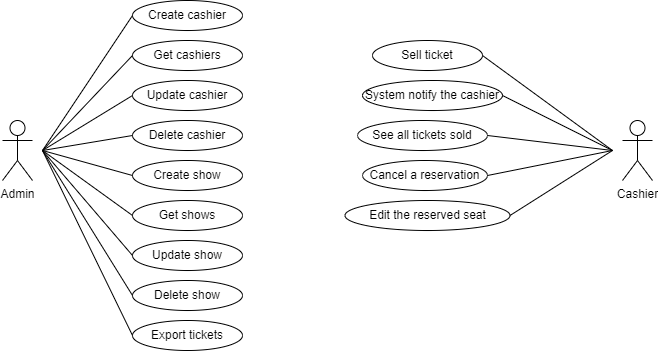
*Level: user-goal level*

*Primary actor: Cashier*

*Main success scenario: login->Buy\_ticket->enter show id, seat number, seat row->ticket is added to database*

*Extensions: login -> incorrect username or password,*

*login -> enter show id, seat number, seat row -> the ticket with this seat number and seat row already exists*

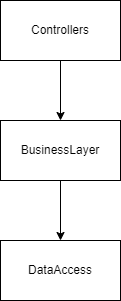


3. System Architectural Design

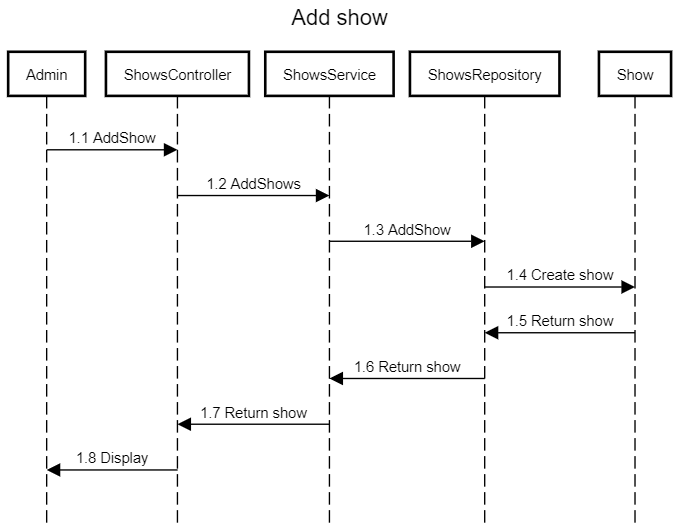
**3.1 Architectural Pattern Description**

*The architectural pattern used in this application is Layers. Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data (e.g., calculate values or aggregate data), and pass that information up to the presentation layer.*

**3.2 Diagrams**

**

4. UML Sequence Diagrams

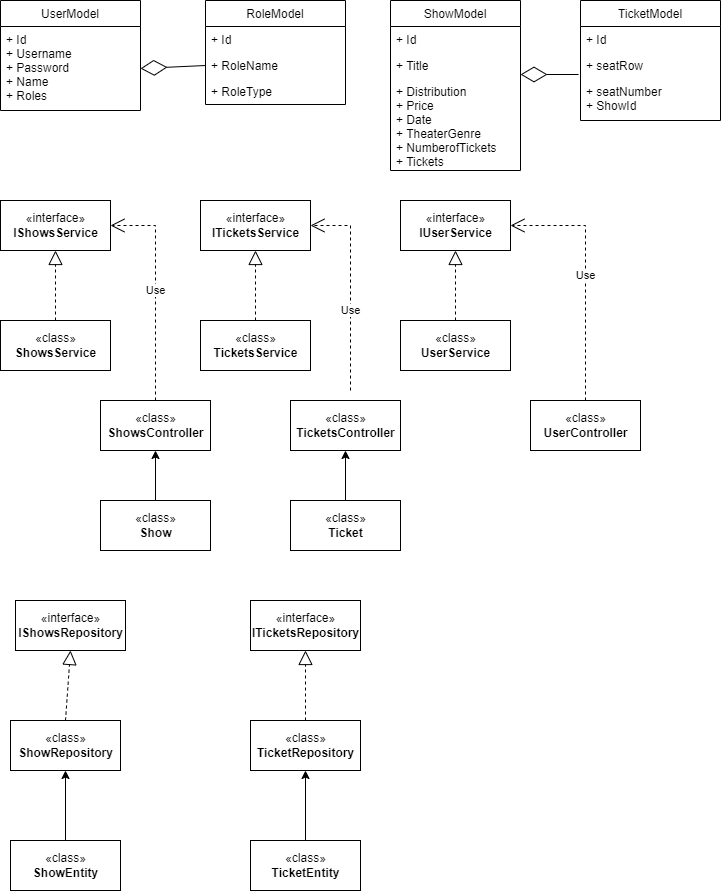


5. Class Design

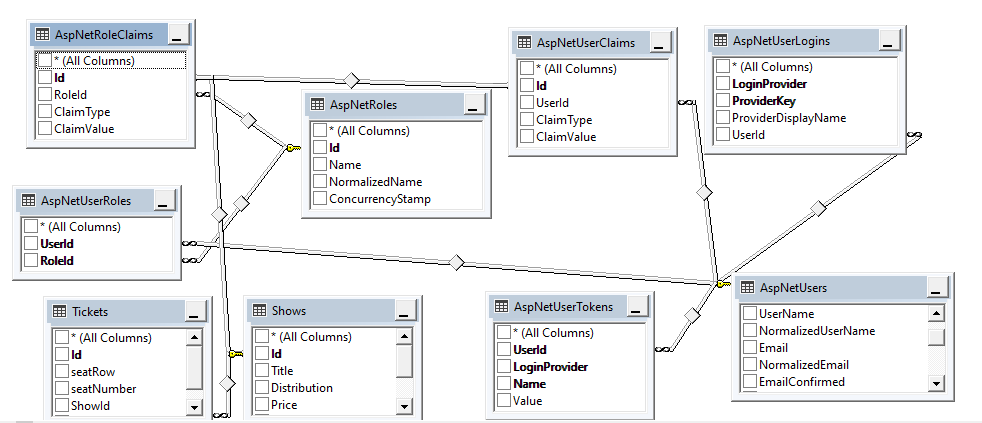
**5.1 Design Patterns Description**

The design pattern used for generating csv file is the Factory method pattern. In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method - either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes - rather than by calling a constructor.

**5.2 UML Class Diagram**

**

6. Data Model

**

7. System Testing

8. Bibliography

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-6.0&tabs=windows