Untold App

Analysis and Design Document

Student: Ștefan Ciuprina

**Group: 30434**

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

The purpose of this assignment is to design and implement a ticket selling system for the Untold festival. The application has two types of users (a cashier and an administrator) which provide a username and a password to use the application.

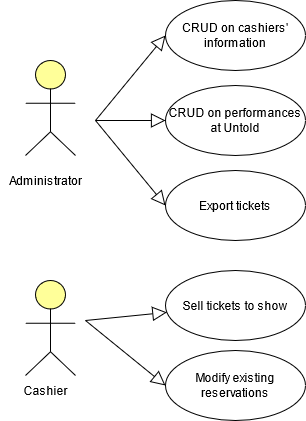
# Functional Requirements

The system designed comprises of the user interface (UI), the client side, in which the user does its job it is supposed to do; namely, the administrator will manage the cashiers and the shows, while the cashier will sell tickets or modify existing ones. Those will be sent as request from the UI to an API handling them, the server side, which will take care of managing the request and communicating with the database.

# Non-functional Requirements

The system has a high usability, the design being intuitive and easy to understand for the user, and with a high speed of completion of each action. The application also has a high level of security, each user having a username and a password, a password which is secured through hashing in the database.

2. Use-Case Model



Use case: CRUD on cashiers’ information

Level: user-goal level

Primary actor: Administrator

Main success scenario: Modification of the cashiers’ database table

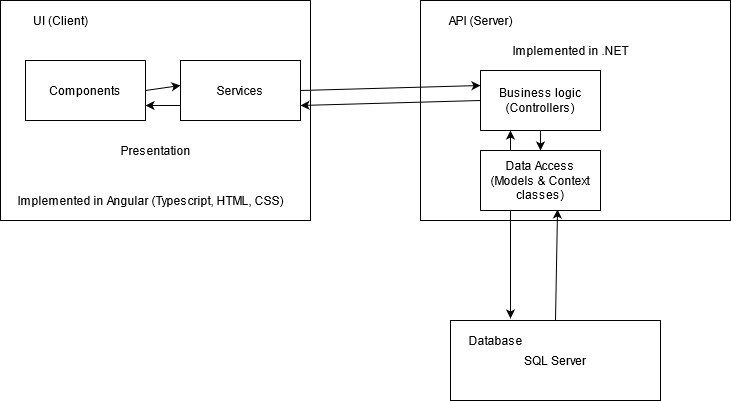
3. System Architectural Design

**3.1 Architectural Pattern Description**

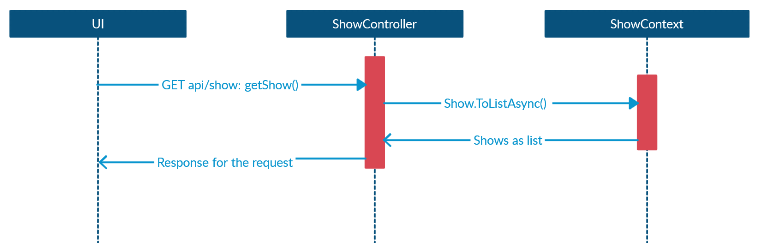
The layered architectural pattern was used for the organization of this application. We have separated the three main components of the app: the data access layer and the business layer are parts of the API (The UntoldApp folder), while the presentation layer is part of the UI (the Untold UI folder). The data access layer and the business layer are separated within the API. The data access is represented in the Models folder, where we have the specific model for each table and their context, connecting the classes to the database. The business layer can be found in the Controller classes, which are using the data access layers to process users’ request. In the presentation layer, we have the user interface, where the user can interact with the API in a pleasant way.

The API was designed in C# using .NET, while the UI was designed in Angular.

**3.2 Diagrams**



4. UML Sequence Diagrams



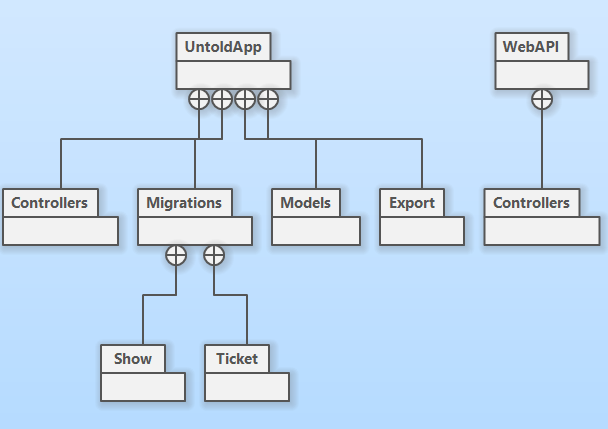
5. Class Design

**5.1 Design Patterns Description**

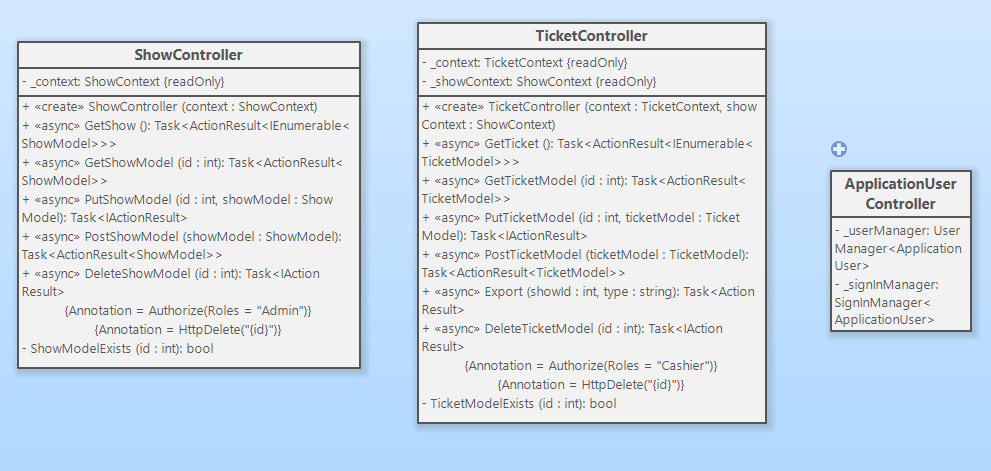
A design pattern used in this application is the Factory Design Pattern, having the purpose of generating the right class for the export of the tickets as either a csv or a xml file, depending on the user’s choice. The Factory Design Pattern prevents the direct creation of a class using the new keyword and creates the object using a method, which decides based on the argument which instance should be returned.

**5.2 UML Class Diagram**

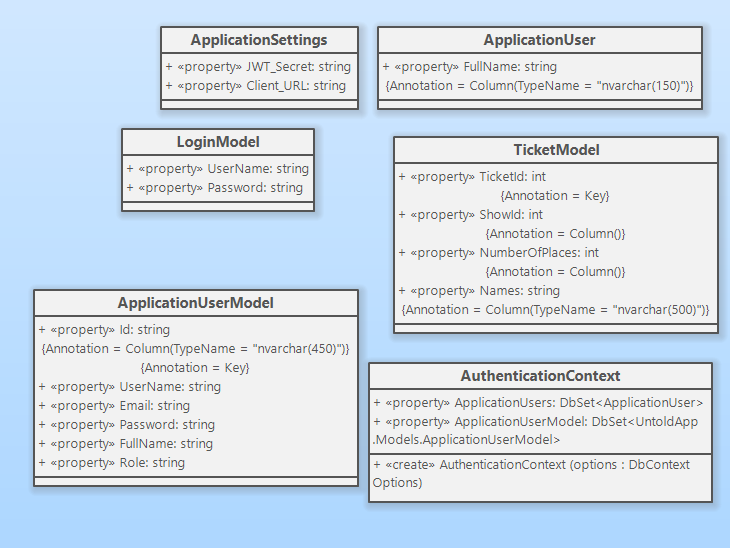
Packages:

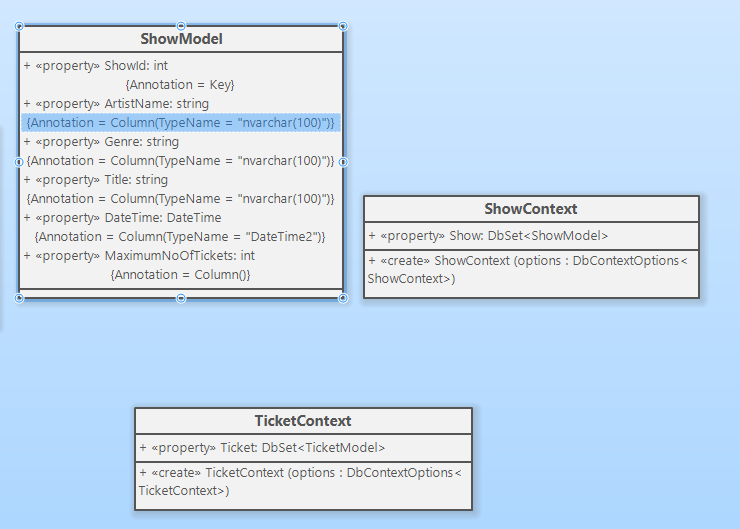


Controllers:

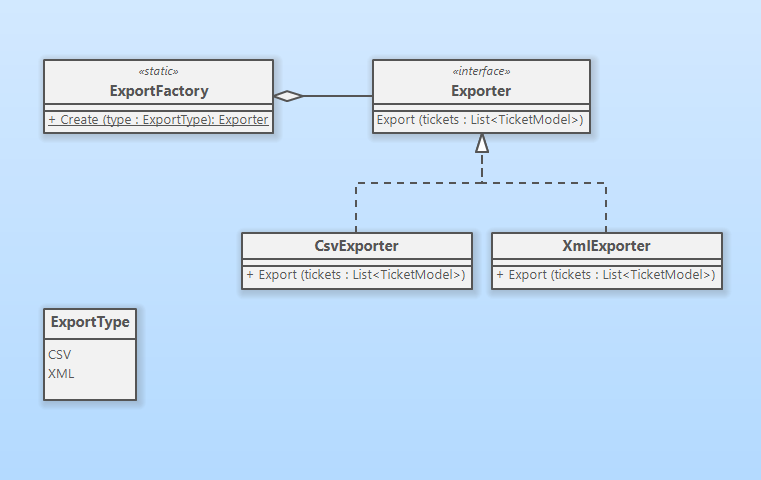


Models:





Export:



6. Data Model

The application uses a SQL Server database for storing the data. The models for authentication have been automatically generated by the ASP.NET Identity, containing information regarding each user (such as username, password, email, full name). The models I have created for this projects are the Show and the Ticket models, which have the fields as shown below:

Show:

-ShowID

-ArtistName

-Genre

-Title

-Date&Time

-Maximum Number of tickets

Ticket:

-TicketID

-ShowID

-NumberOfPlaces

-Names

7. System Testing

The system has been tested in Swagger (for each request) as well as in the UI itself, the application behaving accordingly. Each feature of the use case presented in section 2 is working properly. Validations have also been tested, for example: the user is not able to enter an invalid date for a concert, cannot enter a non-numeric value for the number of tickets and the cashier is notified if it wants to sell a more tickets than there are available.

8. Bibliography

Resources used for developing this project:

<https://jwt.io/> - for security tokens

<https://getbootstrap.com/> - for UI design

<https://swagger.io/> - for http request testing