Assignment 1

Student: Sicobean Alexandra Maria

**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

The objective of this assignment is to test and implement a ticket selling system for the Untold festival. The chosen framework for the project was .NET, due to its adaptability and general use in the software design domain. The application has two types of users: a cashier and an administrator. Both users will need to provide a username and password to access the application.

# Functional Requirements

The administrator user can perform the following operations:

• CRUD (Create, Retrieve, Update and Delete) on cashiers’ information.

• CRUD on the performances at UNTOLD (Tiesto/ Armin/ Martin Garrix/ …). Keep track of the Genre (Techno, Pop, Rap), Title (One last night in Berlin), Date and time of the show (2021 – 08 – 03 1am) and the Maximum Number of tickets per show (20000).

• Administrators can export all the tickets that were sold for a certain show (either in a csv or json file).

The cashier can perform the following operations:

• Sell tickets to a show. A ticket contains the show and can contain one or more places (I can buy a ticket for me and my brother).

• The system should notify the cashier when the number of tickets per show was exceeded.

• A cashier can see all the tickets that were sold for a show, cancel a reservation, or edit it.

# Non-functional Requirements

The system will be intuitive and simply to use. It will provide security to the users by encrypting the passwords when storing them into the database.

2. Use-Case Model

Use case: Login

Level: User-goal level

Primary actor: User (Cashier or Administrator)

Main success scenario:

1. User clicks on the Login button.
2. System presents the login form.
3. User enters their username and password.
4. System verifies the entered credentials.
5. System redirects the user to the appropriate home screen.

Use case: Sell tickets

Level: User-goal level

Primary actor: Cashier/Admin

Main success scenario:

1. Cashier selects a show to sell tickets for.
2. Cashier enters the number of tickets to be sold.
3. Cashier confirms the sale.
4. System updates the ticket database with the new ticket sale.
5. System notifies the cashier if the number of tickets for the show has been exceeded.

Use case: Export tickets

Level: User-goal level

Primary actor: Administrator

Main success scenario:

1. Administrator selects a show to export tickets for.
2. System displays a form to choose the export format (CSV or JSON).
3. Administrator selects the desired format.
4. System exports the ticket data for the selected show to the selected format. Extensions:

Use case: Delete ticket

Level: Sub-function level

Primary actor: Cashier/Admin

Main success scenario:

1. User selects a ticket to delete.
2. System displays the details of the selected ticket.
3. User confirms the deletion.
4. System deletes the selected ticket from the database.

Use case: Edit ticket

Level: Sub-function level

Primary actor: Cashier/Admin

Main success scenario:

1. User selects a reservation to edit.
2. System displays the reservation details.
3. User updates the reservation details (e.g. adds or removes seats).
4. System updates the ticket database with the modified reservation.

Use case: Delete performance

Level: Sub-function level

Primary actor: Cashier/Admin

Main success scenario:

1. User selects a performance to delete.
2. System displays the details of the selected performance.
3. User confirms the deletion.
4. System deletes the selected performance from the database.

Use case: Edit performance

Level: Sub-function level

Primary actor: Cashier/Admin

Main success scenario:

1. User selects a performance to edit.
2. System displays the performance details.
3. User updates the performance details (e.g. adds or removes seats).
4. System updates the performance database with the modified reservation.

Use case: Create performance

Level: User-goal level

Primary actor: Cashier/Admin

Main success scenario:

1. User selects the “New performance” button
2. User enters the details of the performance.
3. System updates the performance database with the new performance sale.

Use case: Delete cashier

Level: Sub-function level

Primary actor: Administrator

Main success scenario:

1. Administrator selects a cashier to delete.
2. System displays the details of the selected cashier.
3. Administrator confirms the deletion.
4. System deletes the selected cashier from the database.

3. System Architectural Design

**3.1 Architectural Pattern Description**

Building a layered architecture for a .NET web application typically involves separating the application into distinct layers based on their responsibilities. Therefore, in my project I had a presentation, business logic, and data access layer. This approach can help improve code organization, maintainability, and scalability.

Here are the steps I followed to build a layered architecture with .NET web application:

1. Define the layers
2. Create projects for each layer
3. Define interfaces
4. Implement the layers
5. Configure Dependency Injection
6. Define Models: Define models for each layer to represent the data they use
7. Implement Controllers in the presentation layer.
8. Implement Services in business layer
9. Implement Repositories in data layer

**3.2 Diagrams**

4. UML Sequence Diagrams

Diagram

Description automatically generated

5. Class Design

**5.1 Design Patterns Description**

For the view layer, I decided to use an MVC design pattern. The main controller is the HomeController. I have different views for each user and CRUD operation. However, the Model layer is only used for persistent data such as Booleans that determine whether an admin or cashier is logged in or not.

**5.2 UML Class Diagram**

A picture containing text

Description automatically generated

The only proper classes in the project are the ones above. The controller contains an object for each of the interfaces above.

6. Data Model

*Graphical user interface, application

Description automatically generated*

1. System Testing

Creating a performance:

Table

Description automatically generated

Table

Description automatically generated

Delete performance:

Table

Description automatically generated

Edit performance:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Creating a new ticket:

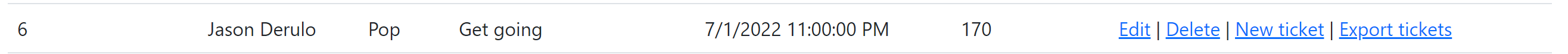
Graphical user interface, text, application, email

Description automatically generated

Chart, scatter chart

Description automatically generated

You can see how the nr of tickets was changed when a ticket to Jason Derulo was bought.



8. Bibliography

1. <https://medium.com/aspnetrun/layered-architecture-with-asp-net-core-entity-framework-core-and-razor-pages-53a54c4028e3>

2. <https://dotnet.microsoft.com/en-us/learn>

3. <https://www.javatpoint.com/net-framework>