Ticket Selling System

Student: Antonescu Maria-Cristina

**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 assignment requires the development of a ticket-selling system for the Untold festival. The system should have two types of users, a cashier and an administrator, who must provide a username and password to access the system. The administrator should be able to perform CRUD operations on the cashiers' information, as well as CRUD operations on the performances at UNTOLD, keeping track of the genre, title, date and time of the show, and the maximum number of tickets per show. Administrators should be able to export all the tickets that were sold for a certain show in either a CSV or JSON file. The cashier should be able to sell tickets to a show, and the system should notify the cashier when the number of tickets per show is exceeded. The cashier should also be able to see all the tickets that were sold for a show, cancel a reservation, or edit it.

# Functional Requirements

*1.2.1. The system should allow the administrator to log in with a username and password.*

*1.2.2. The system should allow the administrator to perform CRUD operations on the cashiers' information, including adding new cashiers, deleting cashiers, updating cashier information, and retrieving cashier information.*

*1.2.3. The system should allow the administrator to perform CRUD operations on the performances at UNTOLD, including adding new shows, deleting shows, updating show information, and retrieving show information.*

*1.2.4. The system should allow the administrator to export all the tickets that were sold for a certain show in either a CSV or JSON file.*

*1.2.5. The system should allow the cashier to log in with a username and password.*

*1.2.6. The system should allow the cashier to sell tickets to a show, including selecting the show, the number of tickets, and the seats.*

*1.2.7. The system should notify the cashier when the number of tickets per show is exceeded.*

*1.2.8. The system should allow the cashier to see all the tickets that were sold for a show, cancel a reservation, or edit it.*

# Non-functional Requirements

1.3.1. Performance: The system should be able to handle a large number of concurrent users and transactions without significant slowdowns or crashes. The response time of the system should be kept at an acceptable level to ensure that users can complete their transactions efficiently.

1.3.2. Security: The system should ensure the confidentiality, integrity, and availability of user data. User authentication and authorization mechanisms should be implemented to prevent unauthorized access to the system. Sensitive information such as passwords and payment information should be encrypted to protect them from theft or misuse.

1.3.3. Usability: The system should be user-friendly and intuitive, with clear and concise instructions provided to users. The user interface should be responsive and easy to navigate, with relevant information presented in a visually appealing manner.

1.3.4. Availability: The system should be available to users at all times, with minimal downtime for maintenance or upgrades. Any planned maintenance should be communicated to users in advance, with a clear indication of when the system will be available again.

1.3.5. Scalability: The system should be designed to accommodate future growth and expansion, with the ability to handle an increased volume of users and transactions. The system should be modular and extensible, with well-defined interfaces between components.

1.3.6. Reliability: The system should be reliable and robust, with minimal errors and failures. The system should be able to recover quickly from any failures or errors, with mechanisms in place to ensure data integrity and consistency.

1.3.7. Maintainability: The system should be easy to maintain and update, with clear documentation and well-organized code. The system should be designed to minimize the impact of changes or updates on other components, with a focus on modularity and encapsulation.

2. Use-Case Model

***Use case****: Add new ticket*

***Level****: User-goal level*

***Primary actor****: Cashier*

***Main success scenario****:*

*1. Log-in to application*

*2. Introduce ticket information*

*3. Add ticket successfully*

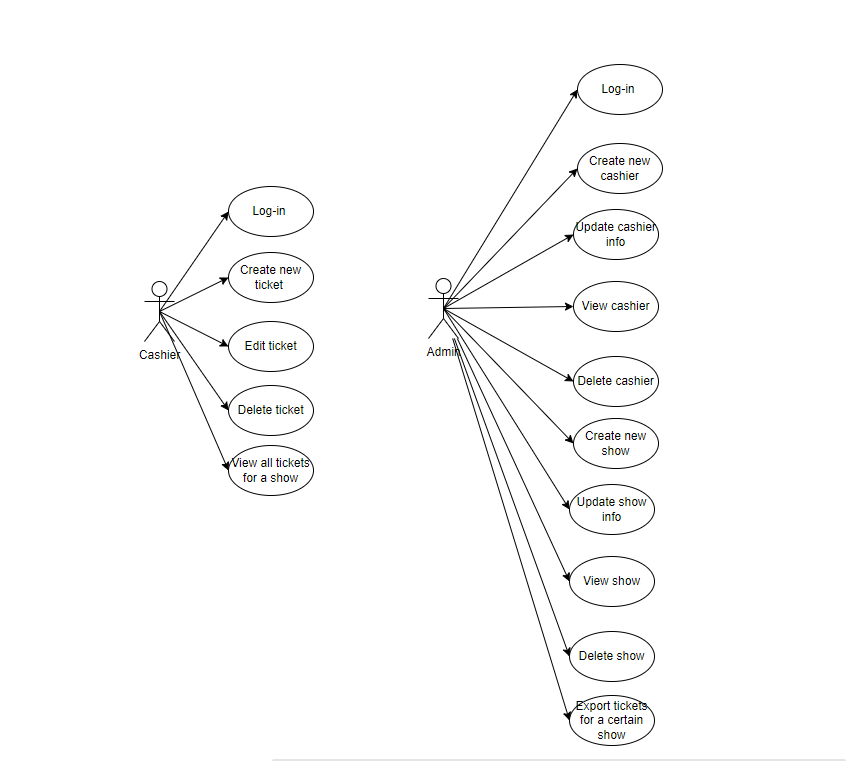
***Extensions****:*

*1. Log-in to application*

*2. Introduce ticket information*

*3. Ticket could not be added because the number of seats exceeds the capacity*

*Use case diagrams:*

**

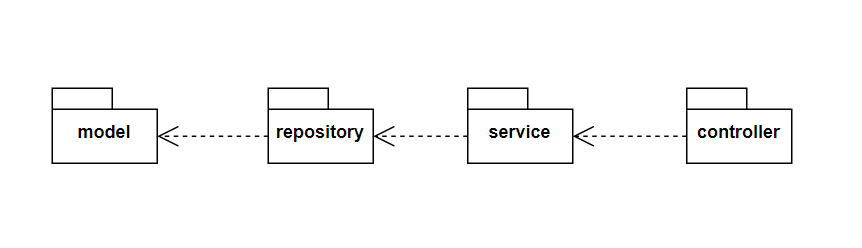
3. System Architectural Design

**3.1 Architectural Pattern Description**

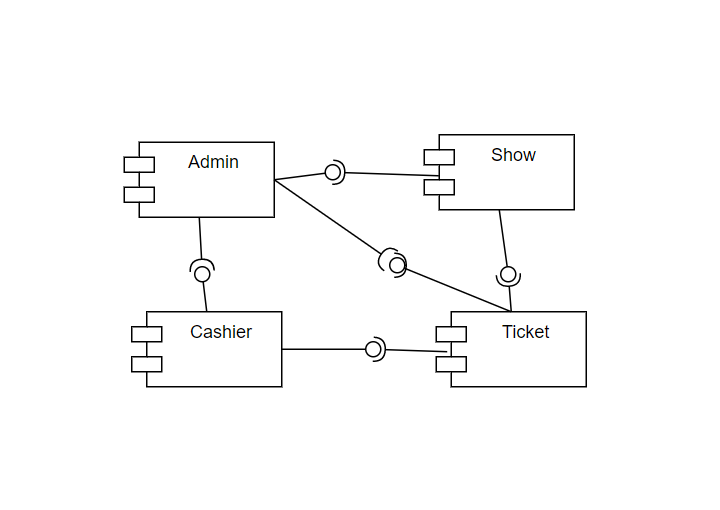
*The application will use the Layers architectural pattern, which separates the application into layers, each with its own set of responsibilities. The layers include the Presentation layer (user interface), Business layer (application logic), and Data Access layer (data storage and retrieval).*

**3.2 Diagrams**

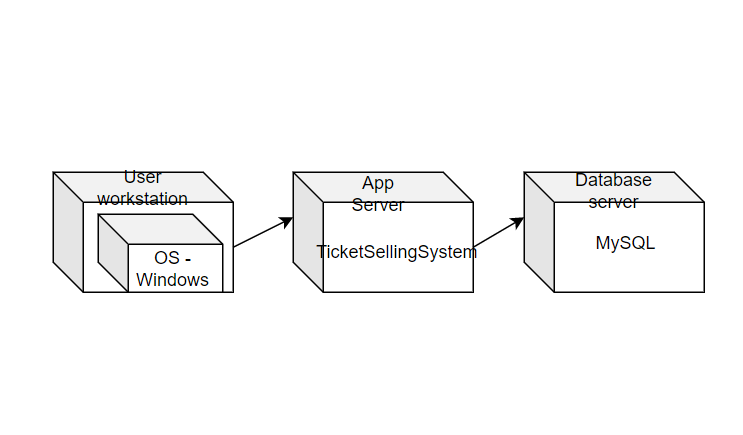
*Package diagram:*

**

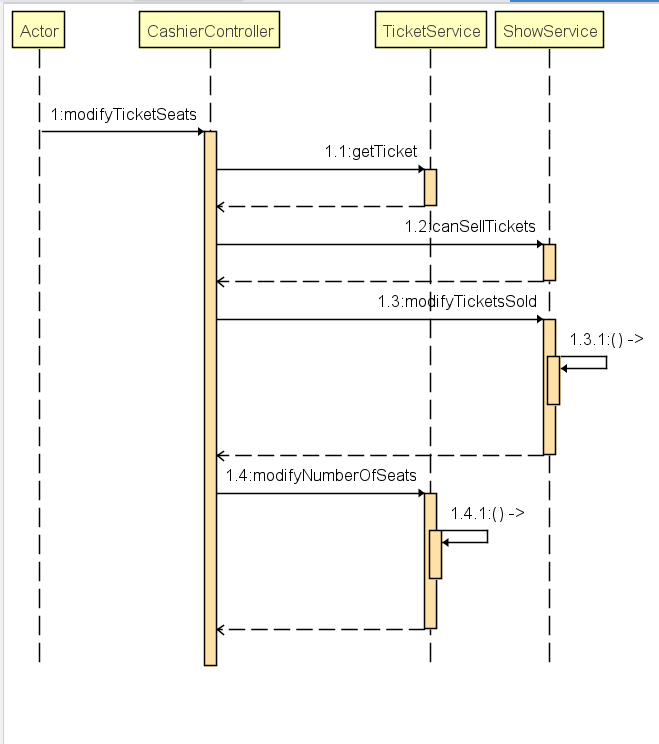
*Component diagram:*

**

*Deployment diagram:*

**

4. UML Sequence Diagrams

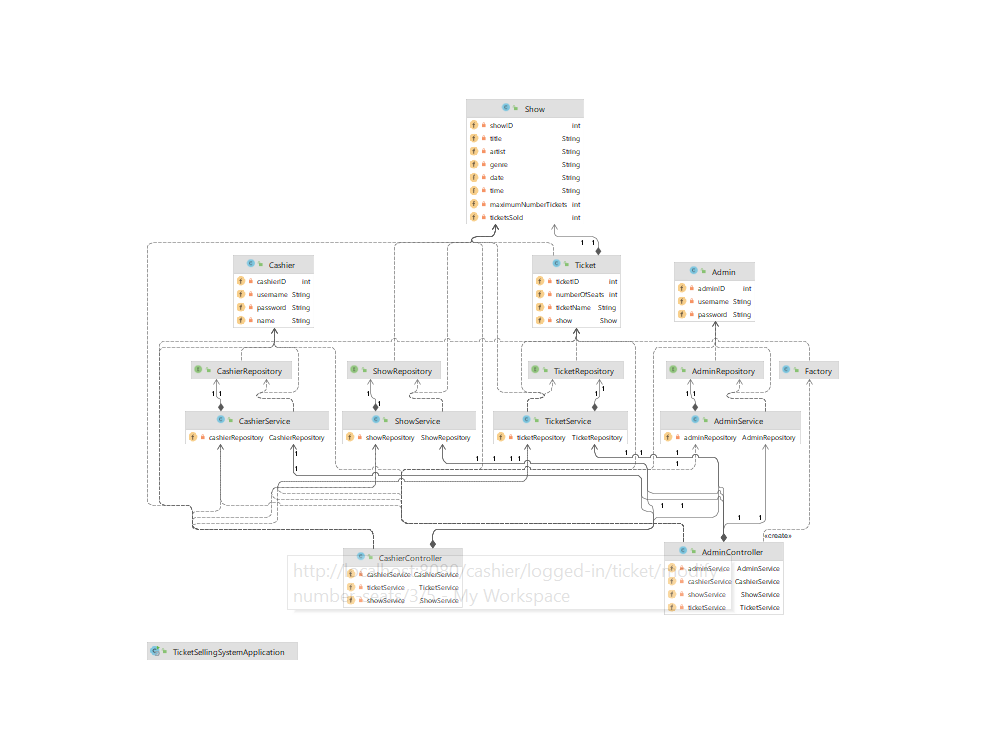
**

5. Class Design

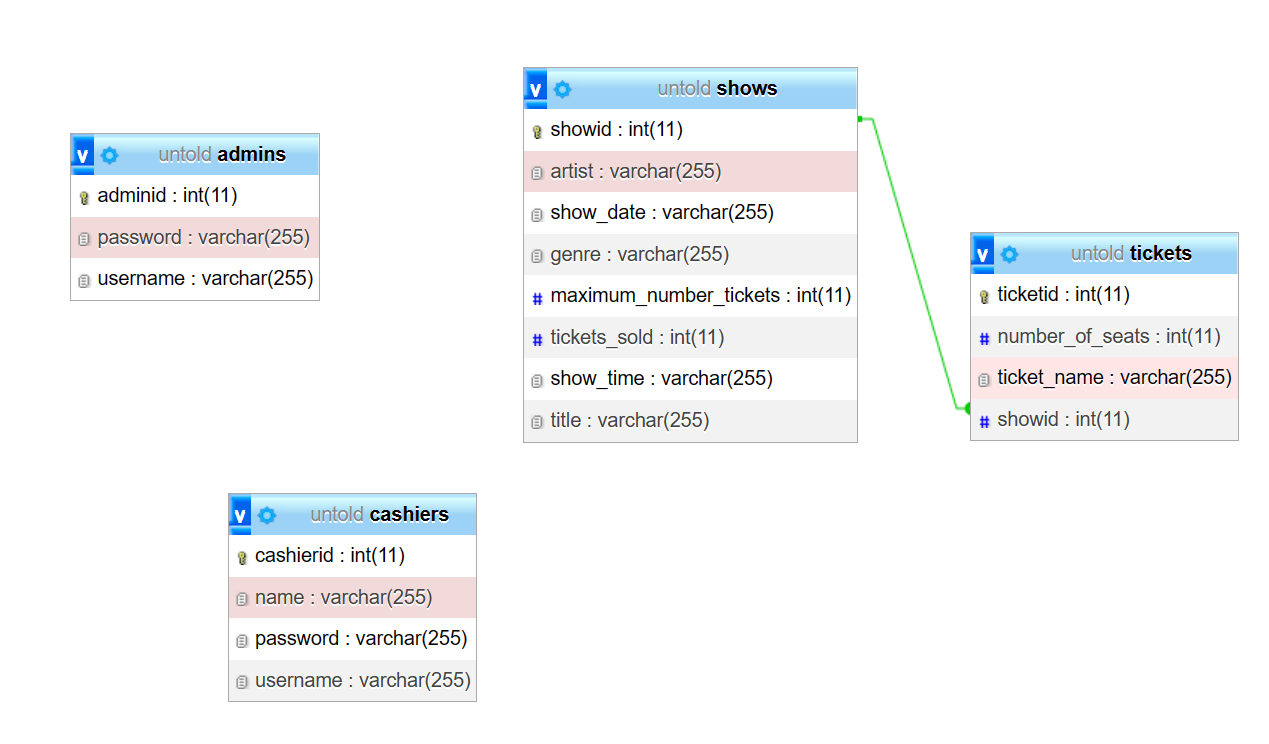
**5.1 Design Patterns Description**

1. *Factory Method Pattern - We will use this pattern to export tickets in either CSV or JSON format. The factory method pattern will allow us to create objects based on the type of export format that we want.*
2. *Repository Pattern - We will use this pattern to interact with the database. The repository pattern will allow us to abstract the data access layer from the rest of the application.*
3. *Service Layer Pattern - We will use this pattern to handle the business logic of the application. The service layer pattern will allow us to abstract the business logic from the rest of the application.*

**5.2 UML Class Diagram**

**

6. Data Model

**

*There are 4 entities in the system:*

* *Admin – is a type of user, which is characterized only by username and password;*
* *Cashier – is the second type of user, which is characterized by username, password and name;*
* *Show – is an entity which describes the name of the concerts, the artist performing them and information such as the time of the concert, the date, the maximum number of tickets available for the show and tickets sold for this show;*
* *Ticket – represents an order for tickets for the shows, described by the name under which the reservation has been done, the number of tickets on the reservation and the show for which the reservation has been done.*

7. System Testing

*For this ticket selling system, we will use the following testing strategies to ensure that the application is functioning correctly. Here are the testing strategies we will use:*

1. *Unit Testing - We will use unit testing to test individual units of code such as methods, functions, or classes. This will help us ensure that each unit of code is functioning as expected and that all edge cases are being handled properly.*
2. *Validation Testing - We will use validation testing to test the input and output of the application. This will help us ensure that the user input is being validated correctly and that the output is correct and meets the system requirements.*

8. Bibliography