<ASSIGNMENT A1
Ticket Selling System>
Analysis and Design Document

Student: Curac Mihai - Ionut
Group: 30431

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

The goal of this project is to design and implement a ticket selling system for a music festival like the Untold festival. The system will have two types of users: cashiers and administrators, each with their respective functionalities. Administrators will manage the users' credentials, the performances, and the shows, while also being able to export tickets' data to csv and xml files. Cashiers will sell tickets and manage them.

1.2 Functional Requirements

The following are the functional requirements of the ticket selling system:

- The system should have two types of users: cashiers and administrators.
- Both types of users must provide a username and a password to access the system.
- Administrators can perform CRUD operations on cashiers' credentials (username and password).
- Administrators can perform CRUD operations on performances at UNTOLD, including on the name, genre, and associated shows.
- Administrators can also perform CRUD operations on shows at UNTOLD, including on the title, date and time, maximum number of tickets, and price per show.
- Administrators can export all tickets sold for a specific show to a csv or xml file.
- Cashiers can sell tickets to a show and manage them.
- The system should notify the cashier when the number of tickets per show was exceeded.
- Cashiers can view all the tickets sold for a show, cancel a reservation, or edit it.
- The system should use a relational database to store all the data.
- The system should use the Layers architectural pattern to organize the application.
- Passwords must be encrypted with a one-way encryption algorithm (Base64).
- The system has unit tests (using JUnit) for the number of tickets per show exceeded validation and the encryption algorithm.
- The system must validate input data, such as the number of seats in a ticket.
- The system must use the factory method to export data to csv/xml.

1.3 Non-functional Requirements

The following are the non-functional requirements of the ticket selling system:

- The system must be easy to use and intuitive for both cashiers and administrators.
- The system must have good performance, even with a large number of users and data.
- The system must be secure and protect user data from unauthorized access.
- The system must have high availability and be able to recover quickly from failures.
- The system must have good scalability to handle increasing user and data loads.

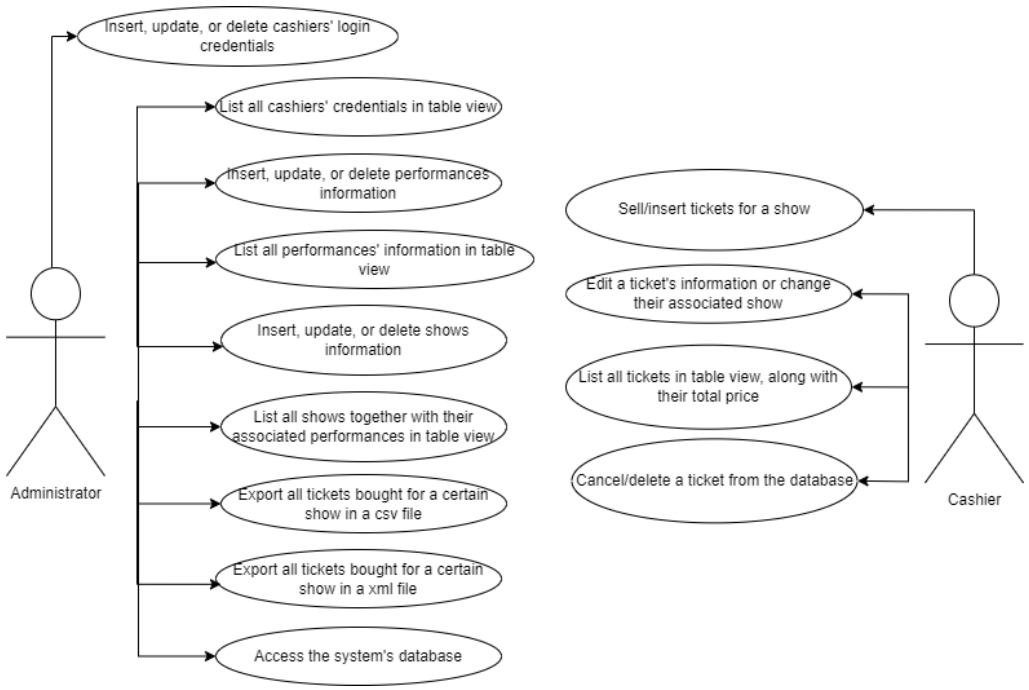
2. Use-Case Model

Use case: Cashier updates ticket information
Level: Sub-function
Primary actor: Cashier
Main success scenario:

1. The cashier enters the ticket's new information.
2. The system verifies that the ticket ID and show ID are correct integer numbers.
3. The system verifies that the show with the given ID exists.
4. The system checks if there are enough seats remaining for the show by comparing the number of seats from the ticket with the ticket number from the show.
5. If the show ID corresponding to the ticket was changed from the previous one, the old show's number of tickets will be updated by adding the original ticket's number of seats, and the new show's number of tickets will be decremented by the new number of seats. If not, the same show's number of tickets will be decremented only by the difference between the original ticket's number of seats and the new number of seats.
6. The ticket will be updated in the database, as well as the shows.
7. The system displays a success message to the cashier.

Extensions:

- If the ticket ID or show ID are not correct positive integer numbers, the system displays an error message to the cashier.
- If the show with the given ID does not exist, the system displays an error message to the cashier.
- If there are not enough seats remaining for the show, the system displays an error message to the cashier.
- If the input is incorrect, the system displays an error message to the cashier.



3. System Architectural Design

3.1 Architectural Pattern Description

For this project, the Layers architectural pattern will be used to organize the application into the following main layers:

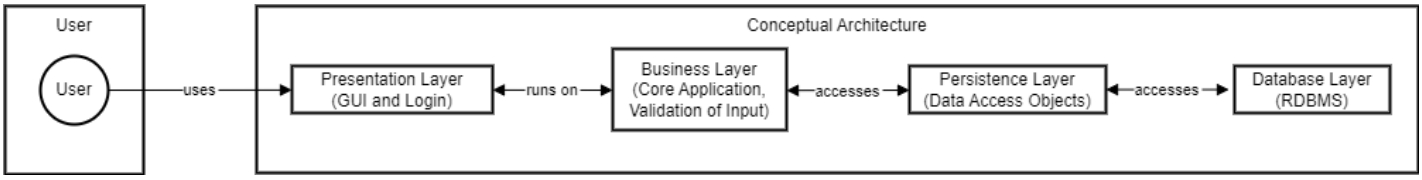
- Presentation layer: responsible for handling the user interface and interaction, including authentication/login.
- Business layer: contains the application's core logic, including ticket sales, updates, and cancelations, CRUD operations on cashiers' credentials, performances, and shows, as well as exporting data to csv and xml files.
- Persistence layer: responsible for communicating with the database and providing data access objects to the business layer.
- Database layer: ensures the connection to the system’s database.

3.2 Diagrams

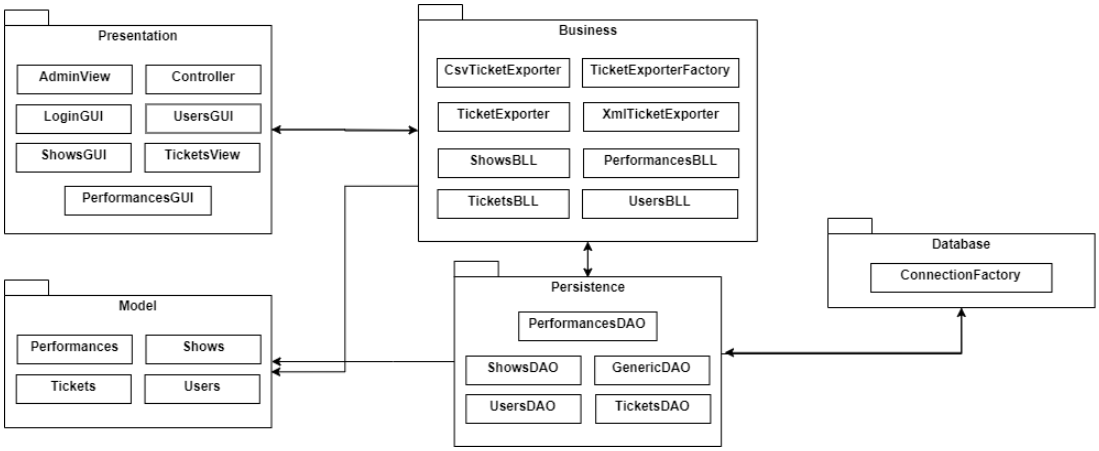
The presentation layer will communicate with the business logic layer to request and send data, which will then be processed by the business layer and stored or retrieved from the persistence layer.

The persistence layer will handle the interaction with the database, including establishing connections, executing queries, and storing data.

This architecture separates concerns and promotes code modularity and maintainability. Additionally, it enables developers to modify specific layers without affecting the overall system.

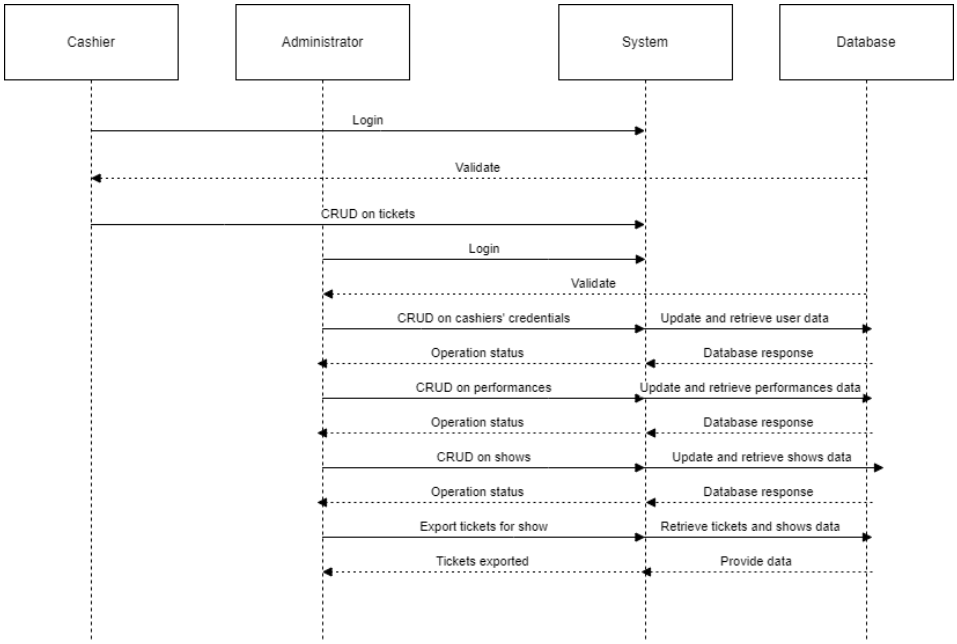


The packages diagram is the following:



4. UML Sequence Diagrams

The sequence diagram for the Tickets Selling System is the following:



5. Class Design

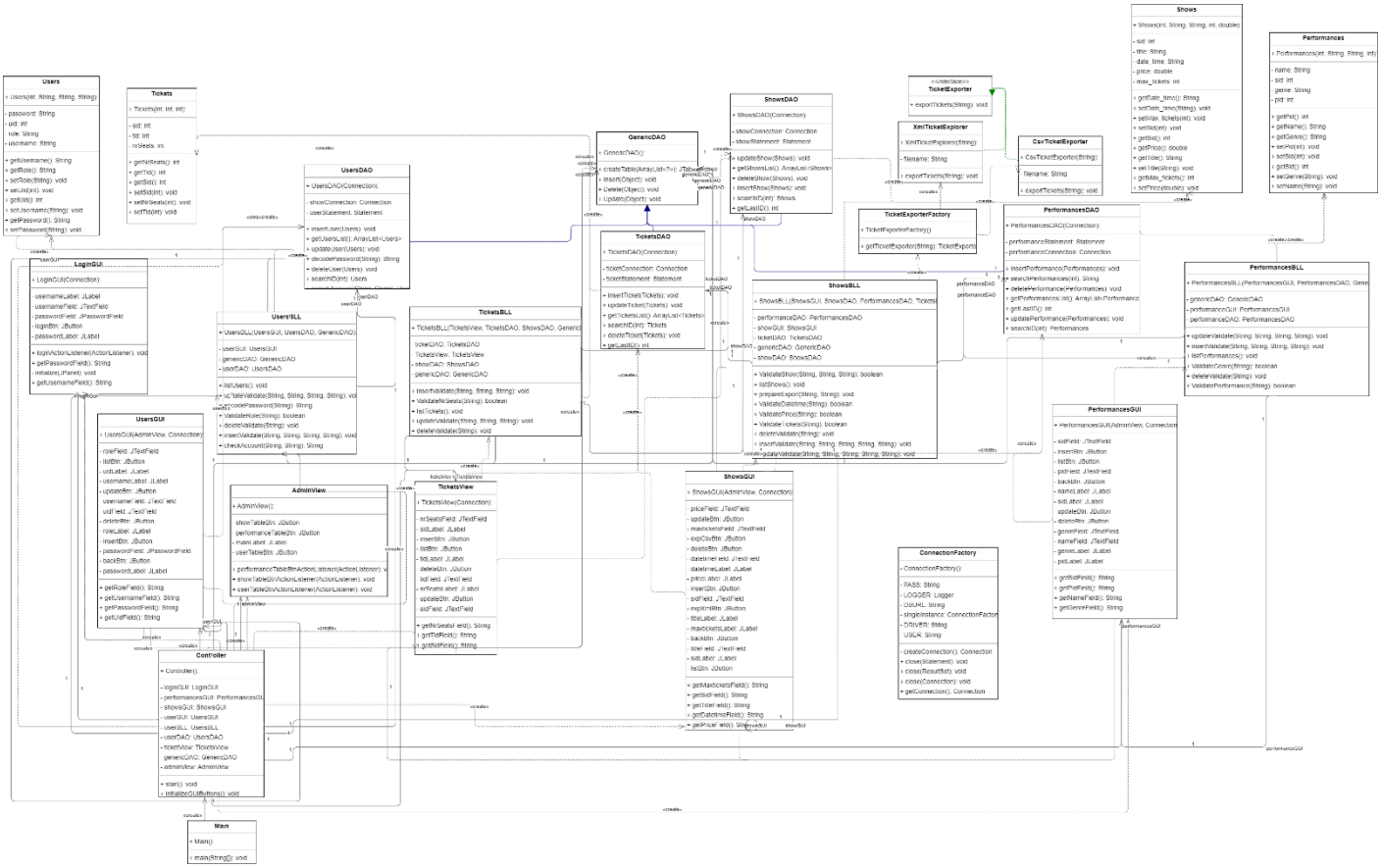
5.1 Design Patterns Description

This project will use the Factory Method Pattern to export data to csv and xml files, as well as the DAO (Data Access Object) Pattern, used for the persistence layer in order to separate the data access code from the business logic.

5.2 UML Class Diagram

The Factory Method Pattern is applied in the TicketExporterFactory class, which has two concrete factories, CsvTicketExporter and XmlTicketExporter. The TicketExporterFactory provides a method to get the specific factory based on the desired file type. The factories implement the TicketExporter interface and each of them overrides the export method, which receives a list of data and writes a string with the formatted data in the correct file.
The DAO Pattern is applied in the Persistence layer, which includes the data access objects for each entity (UsersDAO, GenericDAO, PerformancesDAO, ShowsDAO, and TicketsDAO). These classes handle the database connection and perform the CRUD operations related to their respective entities.

The Class Diagram is the following:



6. Data Model

The data model includes four entities: Users, Performances, Shows, and Tickets, which are represented by their respective classes.

- The Users entity contains the users' id, username, password, and role in the system (cashier or administrator).
- The Performances entity contains the performances' id, name, genre, and corresponding show id.
- The Shows entity contains the shows' id, title, date and time, maximum number of tickets, and price.
- The Tickets entity contains the tickets' id, show id, and number of seats.

7. System Testing

The testing strategies that will be used in the ticket selling system are:

- Unit Testing: This testing strategy will involve testing individual components of the system, such as the encryption algorithm, the number of tickets per show exceeded validation, and the export to csv/xml functionality.
- Integration Testing: This testing strategy will involve testing the integration between different components of the system, such as the interaction between the business layer

and the persistence layer.

- Validation Testing: This testing strategy will involve testing the validity of input data, such as checking that the introduced ticket id and show id are correct integer numbers, and that the show with the given ID exists.

8. Bibliography

- https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html
- https://sajid576.medium.com/factory-pattern-131c6167ea3a
- https://www.baeldung.com/cs/layered-architecture
- https://www.javatpoint.com/design-patterns-in-java
- https://www.baeldung.com/java-dao-pattern
- https://www.baeldung.com/java-factory-pattern
- The information provided in the laboratories.