Ticket Selling System

Student: Zeibel Antonia

**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 goal of this assignment is to implement a ticket selling system for a music festival. The application must have two types of users, a cashier and an administrator, which must provide an username and a password to use the application.

# Functional Requirements

The user can use the application only if the authentication process is successful. The authentication process consists of providing the username and the password to the system. In case of non-existing credentials, the user can register to the system.

After the authentication step, the user can request different operations on the system based on their specific role. For example, the administrator can manage cashiers’ information and festival’s performances. Also, this type of user can export all the tickets that were sold for a certain show. The other type of user, the cashier can perform operations on tickets, like sell a ticket, cancel a reservation or even edit it.

# Non-functional Requirements

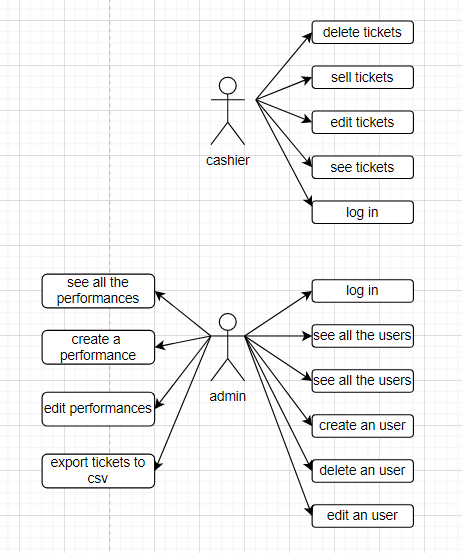
As I have briefly mentioned, the application provides non-functional security features. When the users access their accounts, register or login, the provided details’ are stored into the database, while the password is encrypted for privacy purposes. Moreover, there are specific operations for each type of user according to their role.

There are also validations for these operations. One can observe these validations when creating a ticket or updating one, when updating the information about an user or even a performance.

2. Use-Case Model

**Use-case:** CRUD operations on tickets

**Primary Actor:** Cashier

**Success scenario:** The user logs into the application as a cashier using a valid username and password which were previously created in the registration stage. After the cashier is logged in, the specific operations (create a ticket/sell a ticket, edit a ticket/update a ticket, delete a ticket) can be performed.

**Extensions:** The cashier is notified when there are no more available tickets.

**Use-case:** CRUD operations on cashiers/performances

**Primary Actor:** Admin

**Success scenario:** The user logs into the application as an admin using a valid username and password. After the admin is logged in, the specific operations (create a performance/cashier, edit a performance/cashier, delete a performance/cashier, export tickets into a CSV file) can be performed.

**Extensions:** The admin cannot perform these operations if the login process was not successful.

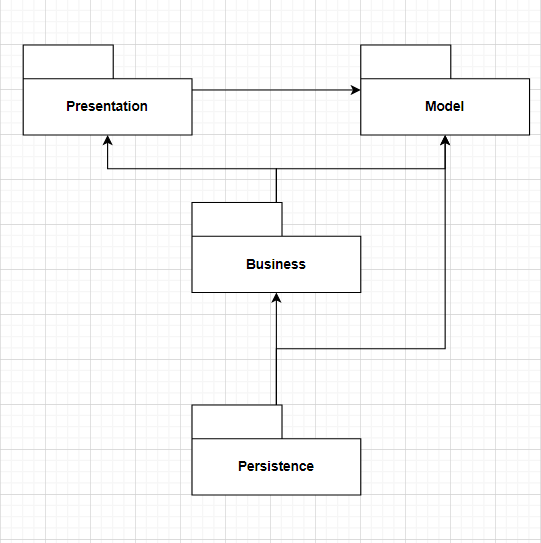
3. System Architectural Design

**3.1 Architectural Pattern Description**

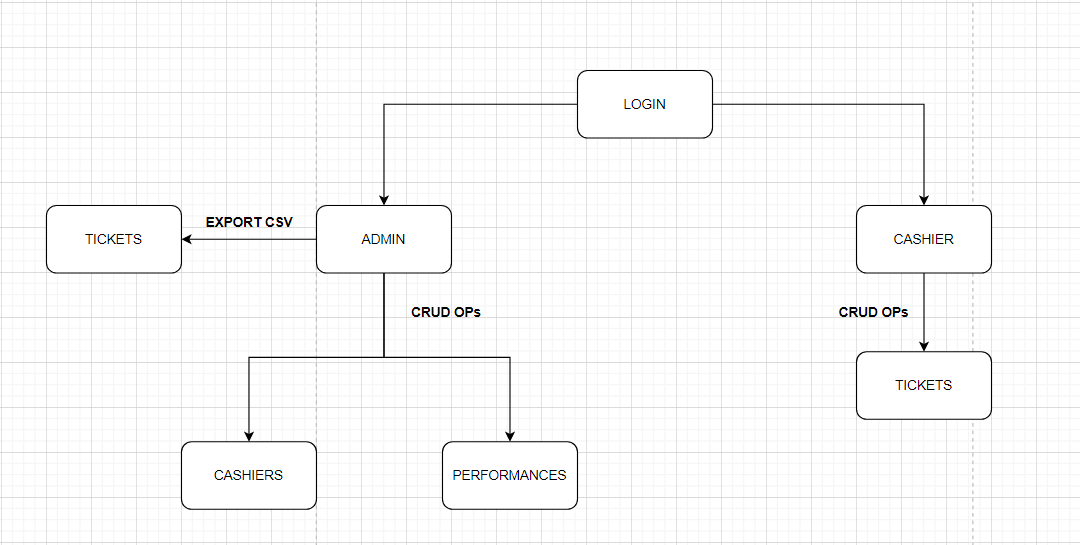
The architectural pattern used for this project is a layered one. This type of architecture respects the concept of layers of isolation which means that changes made in one layer of the architecture do not impact components in other layers and moreover, that each layer is independent of the other layers.

**3.2 Diagrams**

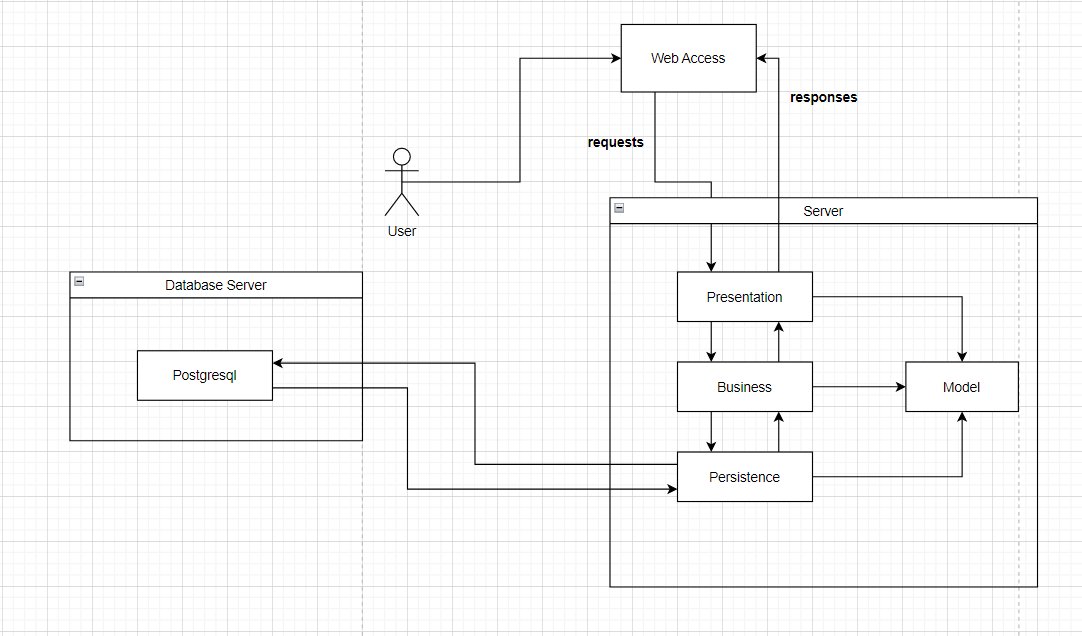
**Package diagram:**

****

**Component diagram:**

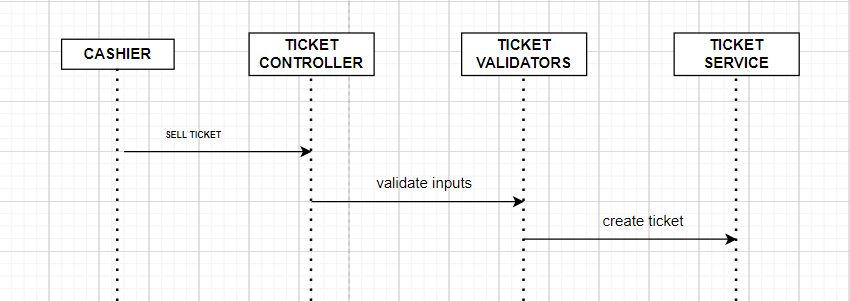
****

**Deployment-diagram:**

****

4. UML Sequence Diagrams

**Sequence diagram for a cashier for adding a new ticket:**

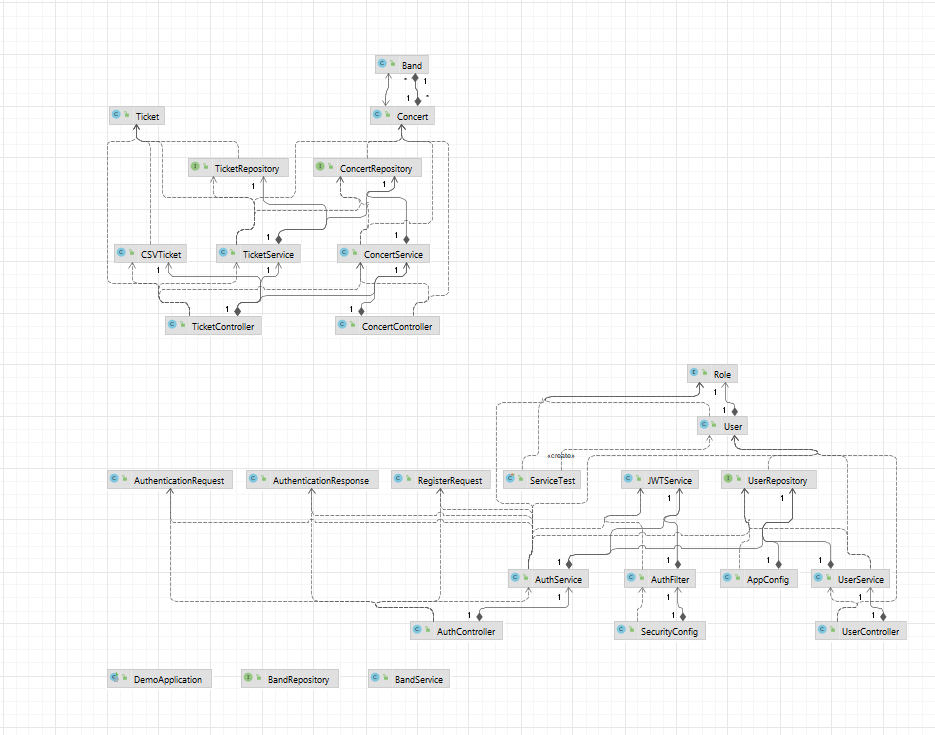


5. Class Design

**5.1 Design Patterns Description**

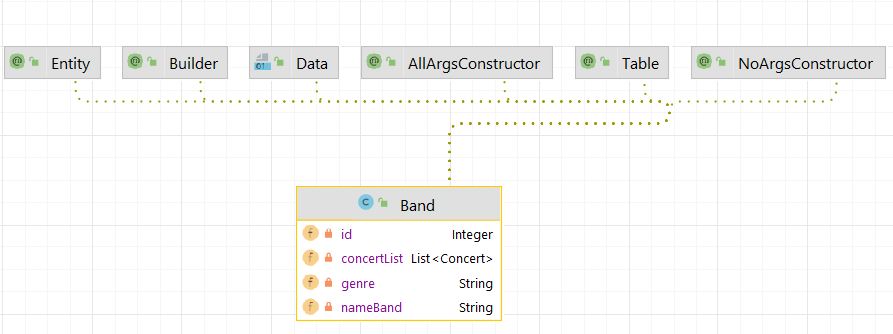
The factory method design pattern is used in this project and it defines an interface or abstract class for creating an object, but gives the decision to the subclasses to decide which class to instantiate. This design pattern allows the sub-classes to choose which type of objects to create. It also promotes the loose-coupling.

**5.2 UML Class Diagram**

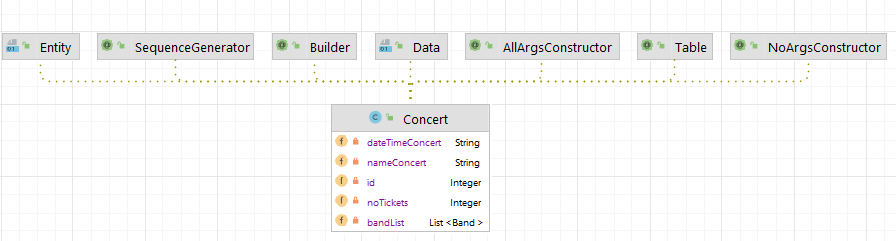
**

6. Data Model

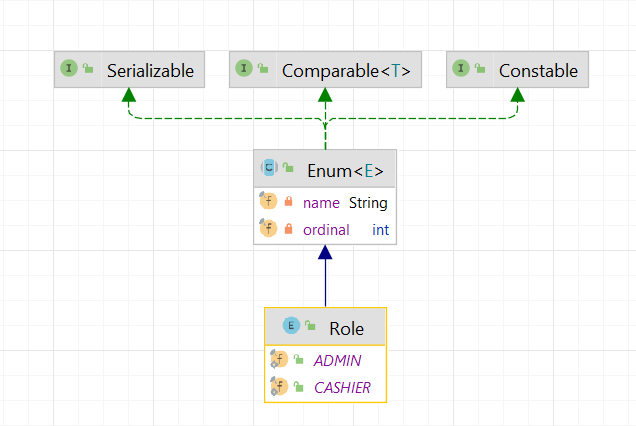
**Band:**



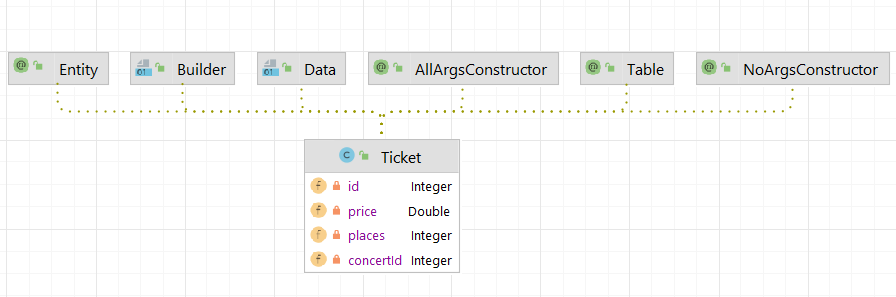
**Concert:**

****

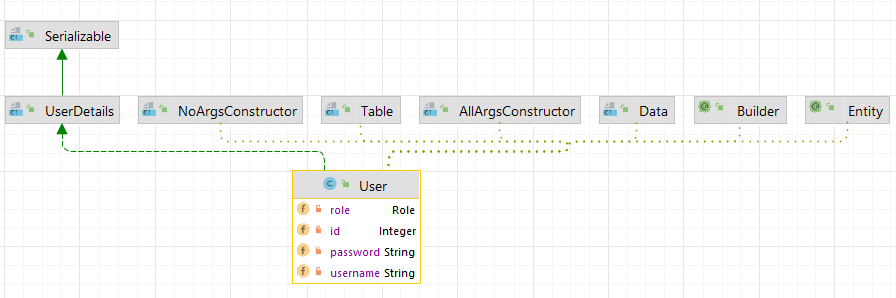
**Role:**

****

**Ticket:**

****

**User:**

****

7. System Testing

For system testing, I have used Junit. A new class named ServiceTest was created where I have tested the method of creating a new user and a new ticket. The @Test annotation was used for both methods and the assertTrue method which return a Boolean value, true if the result of the called method matches the expected result.

8. Bibliography

* Software Design lectures
* <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>
* <https://www.javatpoint.com/factory-method-design-pattern>
* <https://youtu.be/KxqlJblhzfI>
* <https://www.youtube.com/watch?v=9SGDpanrc8U&t=2187s>