<Assignment 1>

Student: Crisan Oana Andra

**Group: 30431/1**

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

This application is the implementation of a ticket selling system. It has 2 types of users: cashiers and administrators. A cashier can sell tickets, modify or delete them and he/she also will be notified when a show is sold out. On the other hand, the admin can create, retrieve, update or delete a cashier or a concert performance. This type of user can also export all the tickets that were sold.

# Functional Requirements

*Here are the functional requirements for the 2 types of users:*

*Admin:*

* *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).*
* *Export all the tickets that were sold for a certain show (either in a csv or json file).*

*Cashier:*

* 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

* Data will be stored in a relational database.
* Use the Layers architectural pattern to organize your application.
* Passwords must be encrypted when stored to the database with a one-way encryption algorithm (base 64).
* Unit tests for the number of tickets for show exceeded validation and the encryption algorithm.
* Use of validations (for example if the number of places in the sell ticket screen is bellow 1).
* Use factory method (not factory) for export to csv/xml.

2. Use-Case Model

***1. Use-Case for Cashier***

*Diagram

Description automatically generated*

*Use case: SELL A TICKET*

*Level: a cashier introduces the data necessary to create a ticket*

*Primary actor: Cashier*

*Main success scenario: ticket is added into the database*

*Extensions: the written fields are not correct, so the ticket is not inserted into the database*

*Use case: EDIT A TICKET*

*Level: a cashier introduces the data necessary to edit a ticket, especially the id*

*Primary actor: Cashier*

*Main success scenario: ticket is edited into the database*

*Extensions: the written fields are not correct or the ticket doesn’t exists, so the ticket is not modified into the database*

*Use case: DELETE A TICKET*

*Level: a cashier introduces the id of the ticket that she/he wants to be deleted*

*Primary actor: Cashier*

*Main success scenario: ticket is deleted from the database*

*Extensions: the id of the ticket doesn’t exist, so the ticket is not deleted from the database*

*Use case: SEE THE TICKETS*

*Level: a cashier sends a request to view all the tickets*

*Primary actor: Cashier*

*Main success scenario: the tickets from the database are shown*

*Extensions: -*

**2. Use-Case for Admin**

**Diagram

Description automatically generated**

*Use case: ADD A CASHIER*

*Level: an admin introduces the data necessary to add a cashier*

*Primary actor: Admin*

*Main success scenario: cashier is added into the database*

*Extensions: the written fields are not correct, so the cashier is not inserted into the database*

*Use case: EDIT A CASHIER*

*Level: An admin introduces the data necessary to edit a cashier, especially the id*

*Primary actor: Admin*

*Main success scenario: cashier is edited into the database*

*Extensions: the written fields are not correct or the cashier doesn’t exists, so the cashier is not modified into the database*

*Use case: DELETE A CASHIER*

*Level: an admin introduces the id of the cashier that she/he wants to be deleted*

*Primary actor: Admin*

*Main success scenario: cashier is deleted from the database*

*Extensions: the id of the cashier doesn’t exist, so the cashier is not deleted from the database*

*Use case: ADD A CONCERT*

*Level: an admin introduces the data necessary to add a concert*

*Primary actor: Admin*

*Main success scenario: concert is added into the database*

*Extensions: the written fields are not correct, so the concert is not inserted into the database*

*Use case: EDIT A CONCERT*

*Level: An admin introduces the data necessary to edit a concert, especially the id*

*Primary actor: Admin*

*Main success scenario: concert is edited into the database*

*Extensions: the written fields are not correct or the concert doesn’t exists, so the concert is not modified into the database*

*Use case: DELETE A CONCERT*

*Level: an admin introduces the id of the concert that she/he wants to be deleted*

*Primary actor: Admin*

*Main success scenario: concert is deleted from the database*

*Extensions: the id of the concert doesn’t exist, so the concert is not deleted from the database*

*Use case: EXPORT THE TICKETS*

*Level: an admin sends a request to export all the tickets in a csv file*

3. System Architectural Design

**3.1 Architectural Pattern Description**

*The layered architecture pattern consists of multiple horizontal layers, with each layer serving a distinct purpose in the application. Although the number and types of layers may vary, the typical layered architecture consists of four standard layers: presentation, business, persistence, and database. However, in some cases, the business and persistence layers are combined into a single business layer, particularly when the persistence logic is embedded within the business layer components. The number of layers in an application may vary depending on the size and complexity of the application.*

*Each layer in the layered architecture pattern has a specific role and responsibility in fulfilling a particular business request. For instance, the presentation layer deals with user interface and browser communication logic, while the business layer is responsible for executing business rules related to the request. The layers in the architecture provide an abstraction around the work required to meet a specific business request. This means that the presentation layer does not have to worry about how to retrieve customer data, it only needs to display the data on a screen in a particular format. Similarly, the business layer does not concern itself with the data formatting or where the data is coming from; its only concern is to retrieve the data from the persistence layer, perform business logic against the data, and pass it up to the presentation layer.*

*Diagram

Description automatically generated*

**3.2 Diagrams**

*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*

*Package Diagram:*

*Diagram

Description automatically generated*

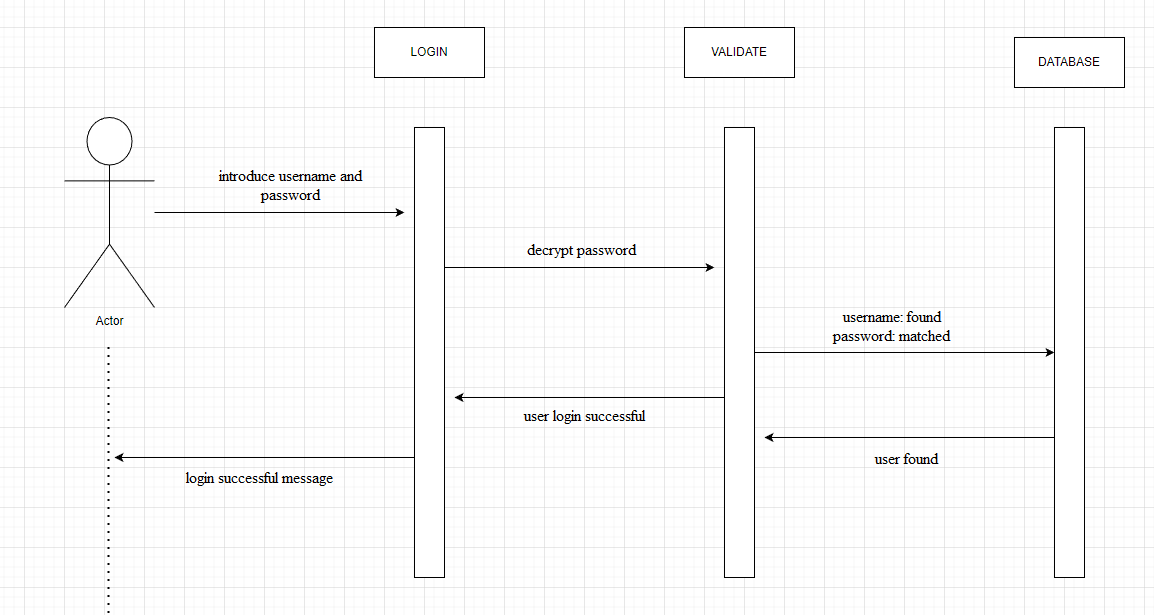
Component Diagram:

Diagram

Description automatically generated

4. UML Sequence Diagrams

*Login scenario for cashier/admin:*

**

5. Class Design

**5.1 Design Patterns Description**

*Repository Design Pattern is commonly used in layered architectures for data access and storage. In the Repository pattern, each data entity or object has a corresponding repository that acts as a mediator between the data source and the rest of the application. The repository provides a set of methods for creating, reading, updating, and deleting data, and it shields the rest of the application from the details of data storage and retrieval.*

*In a Java application with Spring, you might implement the Repository pattern using Spring Data, which provides a set of interfaces and annotations that simplify data access and storage. Spring Data repositories can be easily customized and extended to meet the needs of your application.*

**5.2 UML Class Diagram**

Graphical user interface

Description automatically generated

6. Data Model

*The data models used in this application are Admin, Cashier, Ticket and Concert. Each one of them has the following fields:*

*Graphical user interface

Description automatically generated with medium confidenceGraphical user interface, text, application

Description automatically generated*

*Graphical user interface, application

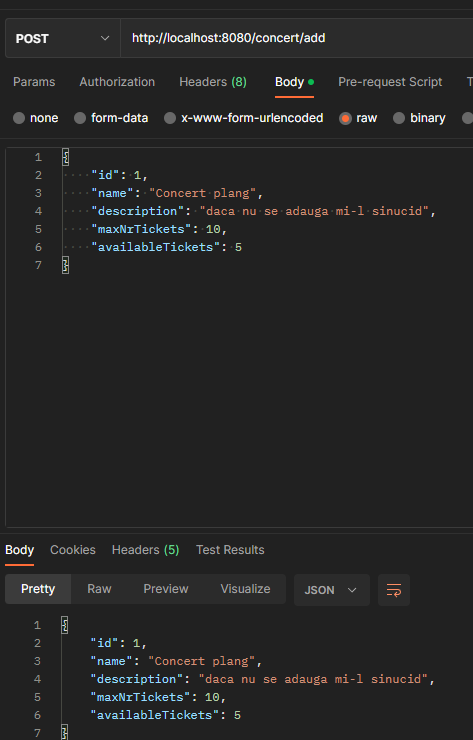
Description automatically generatedGraphical user interface, text, application

Description automatically generated*

7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

*For testing the application I used Postman framework for sending requests and verifying them in the database implemented in Datagrip.*

**

*Graphical user interface, application

Description automatically generated*

*I also implemented Junit tests for password encryption .*

*Text

Description automatically generated*

8. Bibliography

<https://www.jetbrains.com/help/idea/testing.html#add-testing-libraries>

<https://github.com/CristinaMadalinaMihai/UTCNSoftwareDesignLaboratory/blob/main/Assignments/Assignment1_2023.pdf>

* PT courses last year

https://www.youtube.com/watch?v=Gx4iBLKLVHk&t=2983s