Untold Ticket Selling System

Student: Deac Denisa Bianca

**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

Assignment Specification: The objective of this project is to design and implement a ticket selling system using Java and Spring Boot. The system should allow two types of users (a cashier and an administrator) to log in and perform different CRUD operations and other user specific features as the ability to sell tickets for performances at a music festival ,UNTOLD, and track the number of tickets sold for a certain show and export them. The data should be stored in a relational database and the application should follow the Layers architectural pattern, provide encrypted password and validations.

# Functional Requirements

Functional Requirements are represented by the following features of the application:

1. The application allows the cashier and administrator to log in with a username and password.
2. The administrator is able to perform CRUD operations on cashier information, including creating, retrieving, updating, and deleting a cashier from database.
3. The administrator can create a new administrator user.
4. The administrator is able to perform CRUD operations on performances at UNTOLD: for concert and band entities.
5. The administrator is able to export all the tickets sold for a certain show in CSV format.
6. The cashier should be able to perform CRUD operations on tickets: sell tickets for a show, edit number of seats, concert of the ticket, person information, and even cancel a ticket (deleting it).
7. The system notifies the cashier when the number of tickets per show has been exceeded.

# Non-functional Requirements

The Non-functional Requirements of the application are the following:

1. The application is user-friendly and easy to navigate for both cashiers and administrators.
2. The application is responsive and perform tasks quickly and efficiently.
3. The system provides clear and informative error messages for incorrect inputs or other errors.
4. The application provides secure and encrypt passwords in the database with a one-way encryption algorithm.
5. The application follows the Layers architectural pattern, is well-organized and maintainable.

2. Use-Case Model

The following two diagrams represent the visualization of the use cases for each user.

Diagram

Description automatically generated

Diagram

Description automatically generated

* Use case: Edit Ticket

Ticket Level: User-goal level

Primary actor: Cashier Main success scenario:

1. The cashier inputs their correct credentials.
2. The Cashier selects the Edit Ticket option through an http request.
3. The Cashier enters the body of the new Ticket, containing the ID to identify the ticket to be edited.
4. The system retrieves the Ticket details.
5. The system verifies the stock of the Concert and updates the stock accordingly.
6. The system updates the modified Ticket in the database.
7. The system displays a success message to the Cashier with the new Ticket details.

Extensions:

* If the cashier credentials are wrong, the system will display an error message.
* If there are not enough tickets available for the Concert, the system displays an error message.
* If the modification of the Ticket fails for any other reason, the system displays an error message.

3. System Architectural Design

**3.1 Architectural Pattern Description**

A well-liked software design pattern for creating scalable and maintainable programs is the layered architecture pattern, which I chose for my application. This structure separates the application into various layers, each with its own duties. Through clearly defined interfaces, these layers communicate with one another, resulting in a modular and adaptable design.

The Presentation, Business, and Persistence layers make up the three layers that commonly make up the Layered pattern. The user interface and user interaction are handled by the presentation layer. Business logic and data processing are handled by the business layer and the persistence layer is responsible for interacting with the database.

**3.2 Diagrams**

**System Architecture Diagram:**

**Diagram

Description automatically generated**

PERSISTANCE LAYER

The Presentation layer represents the Postman requests, and it's responsible for handling user input and presenting the output. The Business layer consists of controllers and services (that do individual operations) and it's responsible for handling business logic and coordinating data processing. The Model layer contains all the entities needed for the application, including the admin, cashier, ticket, concert, and band. Finally, the Persistence layer contains the repositories for all queries for the database.

**Package Diagram**

All the entities required for the application, including admin, cashier, ticket, concert, and band, are included in the model package. The Model layer of the Layered architecture pattern is represented by this package.

Business package: In the Layered architecture paradigm, this package stands in for the Business layer. Controllers and Services, two of its subpackages, are in charge of managing business logic and coordinating data processing.

All of the controllers that respond to incoming requests from the Presentation layer are included in the Controllers package. (Postman requests). The task of processing incoming requests, calling the necessary services, and providing the client with responses falls on the controllers.

All the services that implement the application's business logic are included in the services package. Each service is in charge of carrying out a particular task.

To access the database, the services communicate with the Persistence layer.

Persistence package: In the Layered architecture pattern, this package represents the Data Access layer. It includes services that perform specific activities and the repository for all database queries. (e.g., export all the tickets that were sold for a certain show in a csv file).

Diagram

Description automatically generated

**Component diagram**

Diagram

Description automatically generated

**Deployment Diagram**

A deployment diagram shows the physical architecture of a system and how the software components are deployed on hardware.

A picture containing diagram

Description automatically generated

4. UML Sequence Diagrams

Sequence diagram for the operation of deleting a concert done by the administrator

*Diagram

Description automatically generated*

5. Class Design

The classes I design follow the architectural pattern in terms of separation into packages and their purpose.

The classes that do the logical manipulation of data are represented by the **Controllers**, one for admin and one for cashier, where the service, methods specific for each use case, are called. The interaction with the user is done via htpp requests, through postman. Each method has a specific path with the necessary annotation for posting, getting, putting or deleting.

The service classes are the one doing the computations and the logical part of the CRUD operations. There is a service class for each entity with specific methods for them. The repositories are instantiated and used here to retrieve the data from the database.

The repositories are the persistance part of the project. I created a repository class for each entity, since these are interfaces that extend JBA Repository, making the connection with database through queries. Some basic query functions are already implemented, but for some entities I made a customized method.

**5.1 Design Patterns Description**

In matters of design patters, the factory method was use as a way of using the Factory Design Pattern. The factory class contains a method that exports all tickets wanted into a csv file and represents a service class for my application.

Text

Description automatically generated

**5.2 UML Class Diagram**

**Diagram

Description automatically generated with medium confidence**

6. Data Model

The data model for my application consists of several entities that represent the main object of the program.

StaffUser is an abstract class that represents a staff member which can be of 2 types: admin and cashier. It has three fields: id of type long, username and password of type String. To implement the differentiation between the 2 admins, I created the class Admin and Cashier that are subclasses of the user.

The Concert class provides fields for id, name, description, dateAndTime, maxTickets, and bands. Using the @GeneratedValue annotation, its id field is created automatically. A concert may feature many bands, as indicated by the @OneToMany annotation in the bands field.

The id field of the Band class is produced automatically. The class also includes name and genre fields.

The price, concert, person, and seats fields in the Ticket class are automatically generated. A ticket can only be connected to one concert, according to the annotation @OneToOne in the concert field.

All these entity classes generate a database of 5 tables as shown below.

A picture containing graphical user interface

Description automatically generated

7. System Testing

The testing of the application is done via postman requests and Junit testing.

Regarding Junit, I had created two functions that test 2 different things: the encryption of the passwords and the action of selling tickets. The encryption follows a successful verification while the selling tickets checks for the error message received while the ticket stock is exceeded.

Text

Description automatically generated

Postman application is also a very useful way to test the ticket selling system by doing http requests. Every use case has a equivalent method with an address for request, but, in order to access the necessary functions, the credential of the user need to be checked. The credentials are the first parameters in each request and are checked each time a function is called such that no unauthorized person uses them.

Graphical user interface, text, application

Description automatically generated

1. Bibliography

* <https://www.youtube.com/watch?v=vOr1ewmCBDw&t=1057s>
* <https://www.youtube.com/watch?v=9SGDpanrc8U>