Assignment 1

Student: Voicu Sara-Ioana

**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 main purpose of this project is to design and implement a ticket selling system for the UNTOLD festival. Relational databases should be used to store the tickets, concert, bands and users. For implementing the desktop application, I choose the Java programming language and the Spring framework.

Other sub-objectives are:

* Analyzing the problem and requirements
* Implementing each task for the users of the application
* Connecting the application to a database and performing different operations on the data
* Sending requests and receiving responses by using the Postman API platform

# Functional Requirements

- Each user has a username, password, and a role

- Any task must be done by a user that logged in

- Allowing only some type of user to have the authority to execute a specific task

- The data from the tables contains concerts, bands, artists, and tickets

- Operations like create, update, delete and view the content should be performed

- Data is stored in a relational database

- Layered architecture pattern is used to organize the application

# Non-functional Requirements

- Using the JSON Web Token that represents a compact and self-contained way for securely transmitting information: useful for authorization, information exchange

- Using encryption algorithms for the passwords of the users

- Helping the users to fill in the mandatory fields, in case of invalid input

- Confirming that the tickets were sold

2. Use-Case Model

Use case: Performing operations on cashiers and performances.

Primary actor: Admin

Main success scenario: The admin logs into the application with valid username and password. Then he can perform CRUD operations on the data of the cashier, which means that he can create a new cashier, update an already existing one, delete a cashier or see a list of all the cashiers. The admin can do the same operations with performances/concerts, that contain information about the bands, date of the concert and number of tickets. He can also generate a .csv file where he can see data about the tickets.

Extensions: If the admin doesn’t introduce the right username and password, he cannot log in.

Diagram

Description automatically generated

Use case: Performing operations on tickets.

Primary actor: Cashier

Main success scenario: The cashier can create an account and then log into the application with valid username and password. Then he can sell tickets to a person, update the content of the tickets, cancel a reservation and see all the tickets available.

Extensions: If we have no more tickets available, then the cashier will be informed and he cannot sell tickets to that concert anymore.

Diagram

Description automatically generated

3. System Architectural Design

**3.1 Architectural Pattern Description**

Layered architectures are said to be the most common and widely used architectural framework in software development. It is also known as an n-tier architecture and describes an architectural pattern composed of several separate horizontal [layers](https://www.baeldung.com/cs/layers-vs-tiers) that function together as a single unit of software. A layer is a logical separation of components or code.

Now, the number of layers in a layered architecture is not set to a specific number and is usually dependent on the developer or software architect. It is important to note that this framework will usually always have a user interaction layer, a layer for processing, and a layer that deals with data processing.

Spring Boot is a module of the Spring Framework. It is used to create stand-alone, production-grade Spring Based Applications with minimum efforts. It is developed on top of the core Spring Framework. Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.

Before understanding the **Spring Boot Architecture**, we must know the different layers and classes present in it. There are **four** layers in Spring Boot are as follows:

* **Presentation Layer:**

The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of **views** i.e., frontend part.

* **Business Layer:**

The business layer handles all the business logic. It consists of service classes and uses services provided by data access layers. It also performs authorization and validation.

* **Persistence Layer:**

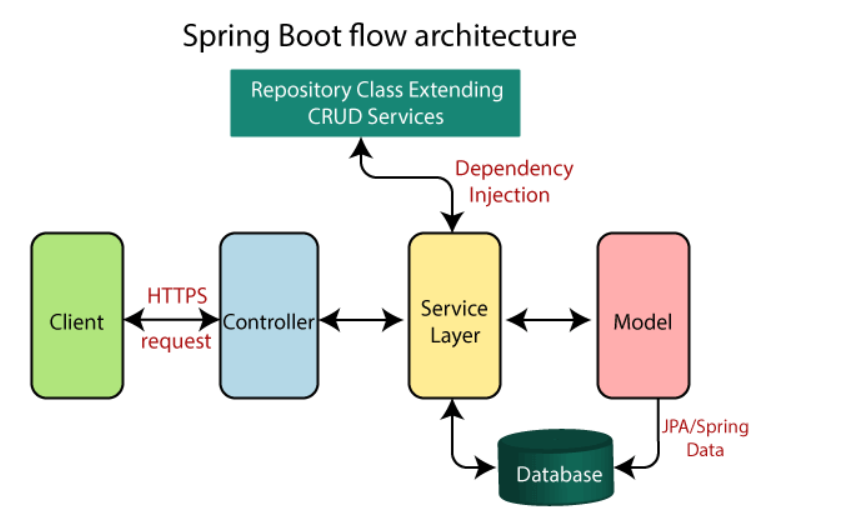
The persistence layer contains all the **storage logic** and translates business objects from and to database rows.

* **Database Layer:**

In the database layer, CRUD (create, retrieve, update, delete) operations are performed.

Diagram

Description automatically generated

**

**3.2 Diagrams**

Component diagram:

Diagram

Description automatically generated

Deployment diagram:

A picture containing text, businesscard

Description automatically generated

4. UML Sequence Diagrams

Diagram

Description automatically generated with medium confidence

5. Class Design

**5.1 Design Patterns Description**

**5.2 UML Class Diagram**

Graphical user interface

Description automatically generated

6. Data Model

**Users:** contains an ID of type Integer, which is a primary key, and username of type String, a password of type String, which is encrypted and a role of type Role, which is an enum and contains ADMIN and CASHIER.

**Concert:** contains an ID of type Integer, which is a primary key, a name for the concert, the number of tickets of type Integer, the date of type String and a list of bands.

**Band:** contains an ID of type Integer, which is a primary key, a name of type String, a genre of type String and a list of Concerts.

**Ticket:** contains an ID of type Integer, which is a primary key, concert id of type Integer, the price of type double, and the number of persons of type Integer.

7. System Testing

For making sure that the application works correctly, JUnit testing was used, where JUnit is a popular unit-testing framework in the Java ecosystem.

For this I created a new class called Test Service, where I have the annotation @Test, which tells the Junit that the public void method in which it is used can run as a test case.

I have 2 tests:

-for the encrypted password

-for the validation of the tickets, to see when the cashier cannot sell more tickets because we have no tickets

I used the method assertTrue() to verify if the application’s result was the expected one.

8. Bibliography

<https://www.youtube.com/watch?v=KxqlJblhzfI>

<https://www.youtube.com/watch?v=9SGDpanrc8U&t=3144s>

<https://www.youtube.com/watch?v=bxy2JgqqKDU&t=5364s>

<https://stackoverflow.com/>

<https://www.baeldung.com/spring-boot-data-sql-and-schema-sql>

<https://www.geeksforgeeks.org/spring-boot-crud-operations/>

<https://springjava.com/spring-boot/export-data-into-csv-file-in-spring-boot>