Student : Tusa Laurentiu

**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 4

4. UML Sequence Diagrams 6

5. Class Design 6

6. Data Model 7

7. System Testing 8

8. Bibliography 9

1. Requirements Analysis

# Assignment Specification

The ticket selling application is a system designed to simplify the process of buying and selling tickets to concerts, through which users interact with tickets to different concerts. It has different access rights for two different types of users ( Admin and Cashier ) to efficiently manage the ticket selling process.

# Functional Requirements

* Two types of users ( Admin and Cashier ).
* Store passwords in an encrypted manner
* Implementation of tests
* Data stored in a relational database

Admin side:

* Export to CSV information about all the tickets to a specific concert
* CRUD ( create, retrieve, update, delete ) operations on all the artists involved
* CRUD ( create, retrieve, update, delete ) operations on all the users involved
* Keep track of concerts dates and maximum number of tickets per concert.

Cashier side:

* Sell tickets to a show
* 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

* Fast response
* User interface
* Use efficient algorithms for encryption
* Memory efficient database
* Accurate tests
* Simple to use
* Prompt the user with steps
* Prompt the user with indications
* Prompt the user with warnings

2. Use-Case Model

O imagine care conține diagramă

Descriere generată automat

Use case: < Sell tickets >

Level: < user-goal level >

Primary actor: < Cashier >

Main success scenario:

1. Log in as cashier
2. Select sell tickets operation
3. Input the buyer’s name
4. Input the amount
5. Select the desired concert
6. The request passes through a verification regarding the limit of available tickets
7. The tickets are bought

Extensions:

1. Log in fails, wrong input is provided, no match with the database. Go back to 1
2. Amount is not an integer. Go back to 3
3. Concert does not exist, wrong input provided. Go back to 5
4. Not enough tickets left . Go back to 6

3. System Architectural Design

**3.1 Architectural Pattern Description**

The layered architecture pattern is a software architecture pattern that divides an application into a set of layers, where each layer provides a specific set of functionality and interacts only with the adjacent layers. This pattern is widely used in software development to achieve separation of concerns, maintainability, and scalability.

The layered architecture pattern typically consists of three or more layers:

Presentation Layer: This layer is responsible for presenting data to the user, usually through a user interface (UI) component. It interacts with the business logic layer to retrieve data and execute business rules.

Business Logic Layer: This layer contains the business logic of the application, which represents the rules that govern how the application functions. It interacts with the data access layer to retrieve and persist data. It represents the layer where the different types of users are identified.

Persistence Logic Layer: This layer is responsible for all the logic behind the operations of request, create, update, delete, export etc. Verifications for the number of tickets available is also computed here.

Data Access Layer: This layer is responsible for accessing the data storage system, such as a database or a file system. It interacts with the business logic layer to provide data to the application and to store data generated by the application.

Additional layers can be added, depending on the specific requirements of the application, such as a service layer for accessing external services or a security layer for implementing authentication and authorization mechanisms.

The benefits of using the layered architecture pattern include better maintainability, testability, and scalability, as each layer can be developed and tested independently. It also promotes separation of concerns, making it easier to make changes to specific parts of the application without affecting the entire system.

**3.2 Diagrams**

O imagine care conține grafic, diagramă

Descriere generată automat

4. UML Sequence Diagrams

*O imagine care conține diagramă

Descriere generată automat*

5. Class Design

**5.1 Design Patterns Description**

The Factory Method Design Pattern was used for the export to CSV operation. It is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

When to use the Factory Method Design Pattern

The Factory Method Design Pattern should be used when:

A class cannot anticipate the type of objects it will need to create

A class wants its subclasses to specify the type of objects it creates

A class wants to defer instantiation of its objects to its subclasses

Provided the fact that there are multiple ways of exporting data from the relational database to external formats (ex CSV, TXT, JSON etc.), the factory method design pattern was used.

How the Factory Method Design Pattern works:

The Factory Method Design Pattern works by defining a Factory Method in a superclass that specifies the signature of the object to be created. The Factory Method can be abstract, which means that the superclass does not provide an implementation for it, or it can be concrete, which means that the superclass provides a default implementation.

Subclasses of the superclass can then implement their own version of the Factory Method to create objects of the appropriate type. Such a subclass was created in this application to implement the export algorithm to a CSV file.

**5.2 UML Class Diagram**

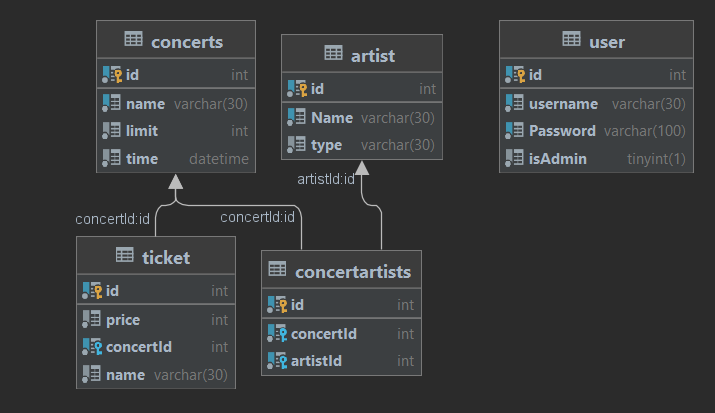
O imagine care conține diagramă

Descriere generată automat

Every table form the database has a class of operations based on one interface. Additional methods are added (ex export tickets for admin only).

Factory method design pattern was used in order to create an instance of a class that implements the interface. An instance of class that executes the export to CSV operation is constructed when the method createExport() is called.

6. Data Model



7. System Testing

JUnit is a popular open-source framework for writing and executing unit tests in Java. It provides a set of annotations, assertions, and test runners that allow developers to easily write and run automated tests for their Java code.

JUnit enables the developers to write test methods that define the expected behavior of their code and use assertions to verify that the actual behavior matches the expected behavior. Test methods can be organized into test suites, and JUnit provides several test runners that can be used to execute these tests and generate test reports.

Some of the key features of JUnit include:

Annotations: JUnit provides several annotations, such as @Test, @Before, and @After, that can be used to mark test methods and set up and tear down methods.

Assertions: JUnit provides a set of assertion methods, such as assertEquals() and assertTrue(), that can be used to verify that the actual result of a test matches the expected result.

JUnit was used in this ticket selling system in order to check the availability of the tickets during a selling operation. The amount requested in evaluated to be greater than 0 first of all, and then to be smaller or equal to the amount of tickets left for a specific concert. The limit of tickets and the amount left is extracted from the database and calculated inside the program’s execution.

O imagine care conține text

Descriere generată automat

8. Bibliography

<https://stackoverflow.com/questions/17538182/getting-keyboard-input>

<https://www.w3schools.com/java/java_switch.asp>

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

<https://app.diagrams.net/>

<https://www.jetbrains.com/help/datagrip/export-data.html#create-a-full-data-dump-for-mysql-and-postgresql>

<https://stackoverflow.com/questions/61536742/mysqldump-with-windows10-os-errno-13-permission-denied-when-executing-sel>

<https://intellij-support.jetbrains.com/hc/en-us/community/posts/360000390819-mysqldump-Path-to-executable-is-wrong>