Assignment A1
Analysis and Design Document

Student:Cristian-Claudiu Chira
Group:30431

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

A ticket management CRUD oriented application(API) that can be used for music venues.

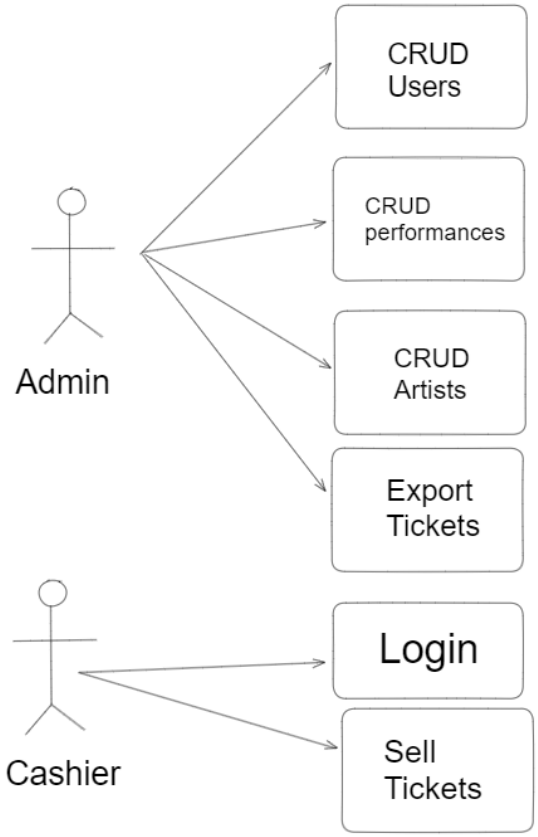
1.2 Functional Requirements

The API should support authentication with two types of users(admins and cashiers) which would then be able to perform CRUD operations on a few types of entities(users, artists, tickets, performances) and also be able to export the sold tickets as a JSON or CSV. It should be built by using a layered architecture(using spring boot here), persisting data(using repositories from JPA and persistence from jakarta)(also this one uses MVC as that's what Spring Boot supports) and also encrypt the user's passwords using Base64.

1.3 Non-functional Requirements

The API should be able to be called by multiple users, and can be instantiated on several ports, all of them updating the same database(ideally, a remote database).

2. Use-Case Model



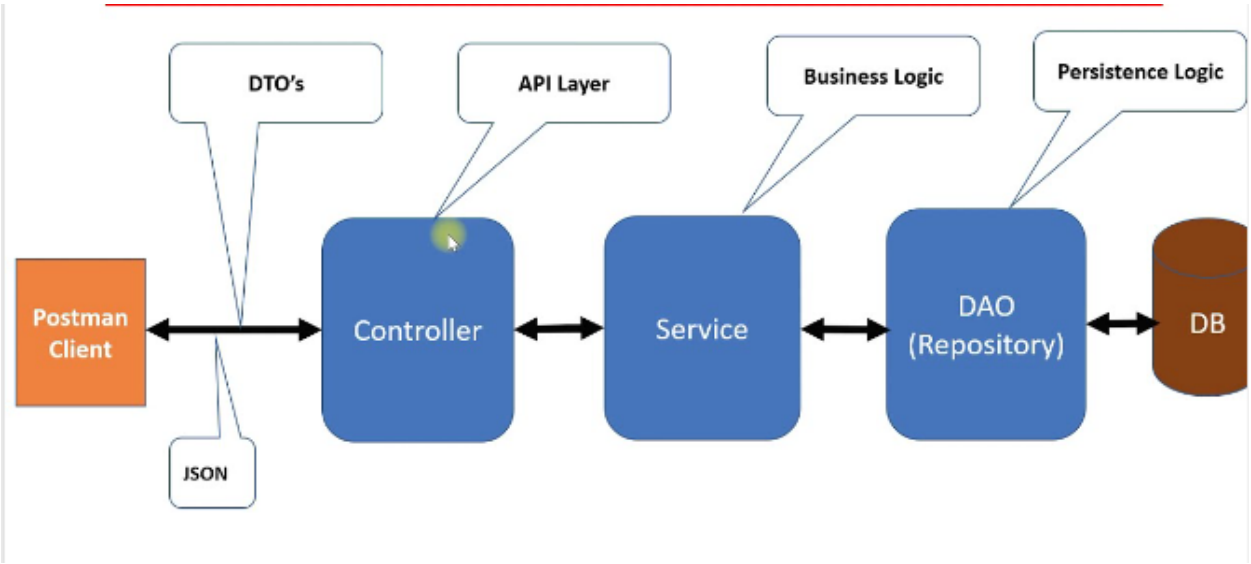
Use case: Export ticket
Level: user-goal
Primary actor: Admin
Main success scenario: Ticket information is retrieved from the database and a JSON file is created containing the serialised data.
Extensions: Error messages in case of unsuccessful authentication.

3. System Architectural Design

3.1 Architectural Pattern Description

The MVC architecture is applied in this use-case, but the View is not implemented. The Controller is where the REST endpoints are defined, and then the Model-tied services for each entity do the appropriate operations and mutations on the Model in order to retrieve the desired data or achieve the desired outcome.

3.2 Diagrams



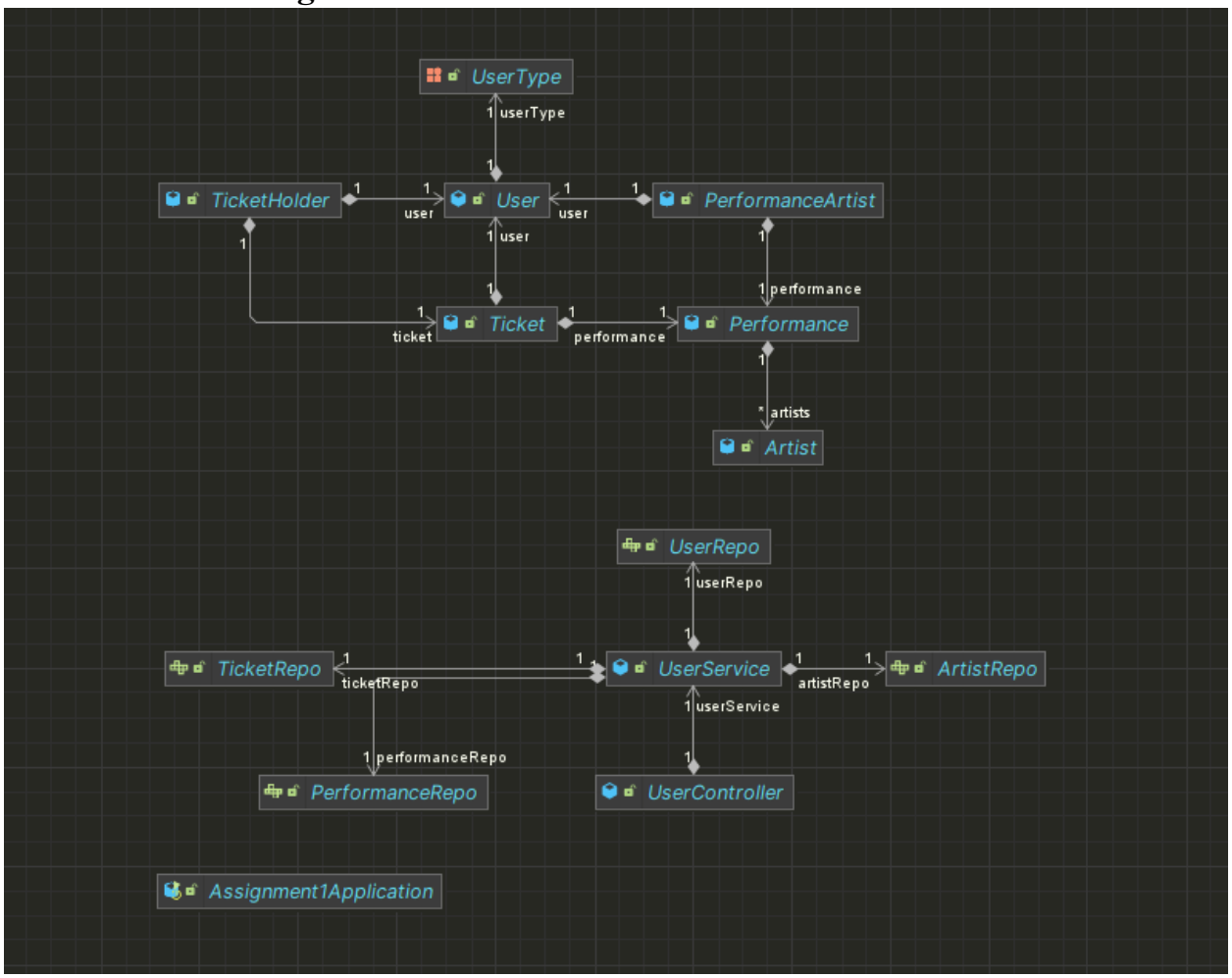
4. UML Sequence Diagrams

5. Class Design

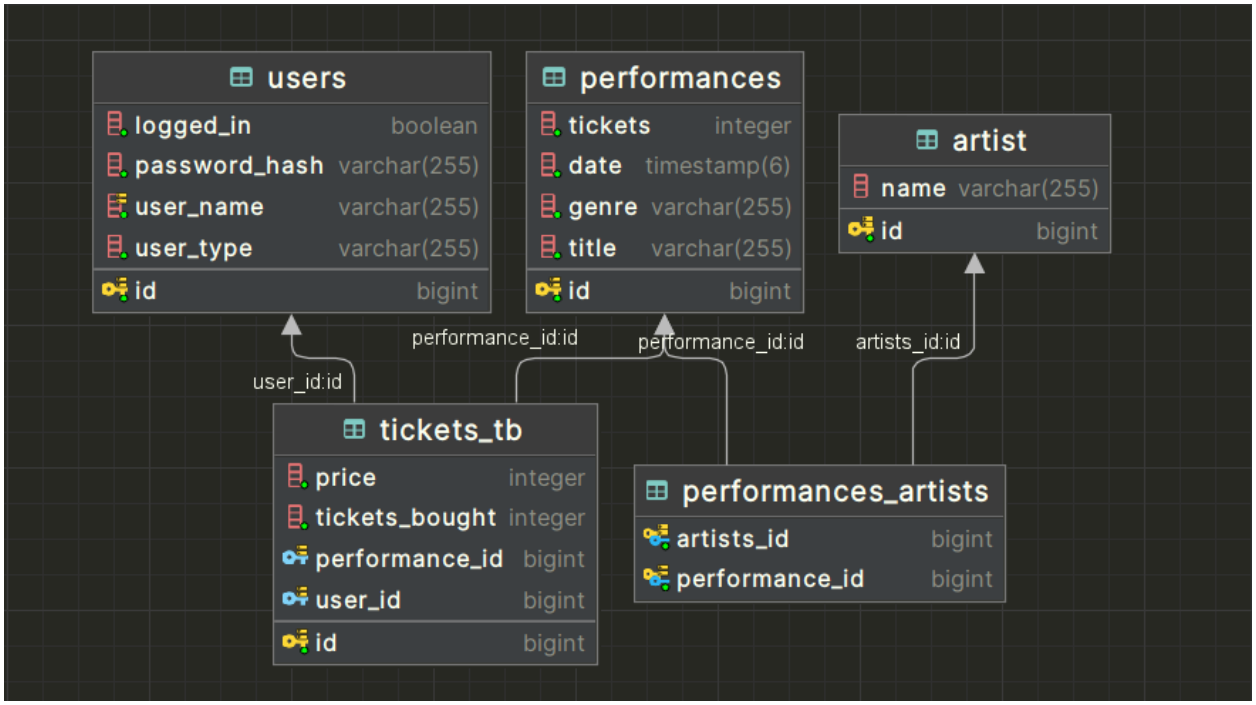
5.1 Design Patterns Description

I am using dependency injection when using - for example - repositories and services. The current class shouldn't be interested in constructing these objects and rather using them. If a differently implemented Repository with the same features comes in but different implementations, it's like an identical puzzle piece made of different material, it fits in perfectly with the rest of the application.

5.2 UML Class Diagram



6. Data Model



7. System Testing

8. Bibliography