Ticket Selling System
Analysis and Design Document

Student: Avram Andrei-Alexandru
Group:30434

Table of Contents

Contents

1. Requirements Analysis

1.1 Assignment Specification

The goal of this assignment was to implement a RESTful API for a Ticket Selling System for Untold Festival with layers architectural pattern, using unit tests and to become familiar with the Factory Method design pattern. The application was created using the Spring Framework and implements the REST API which enables the possibility for this system to communicate with other systems using its endpoints and predetermined data transfer protocols.

1.2 Functional Requirements

The functional requirements this RESTful API system has are the following:

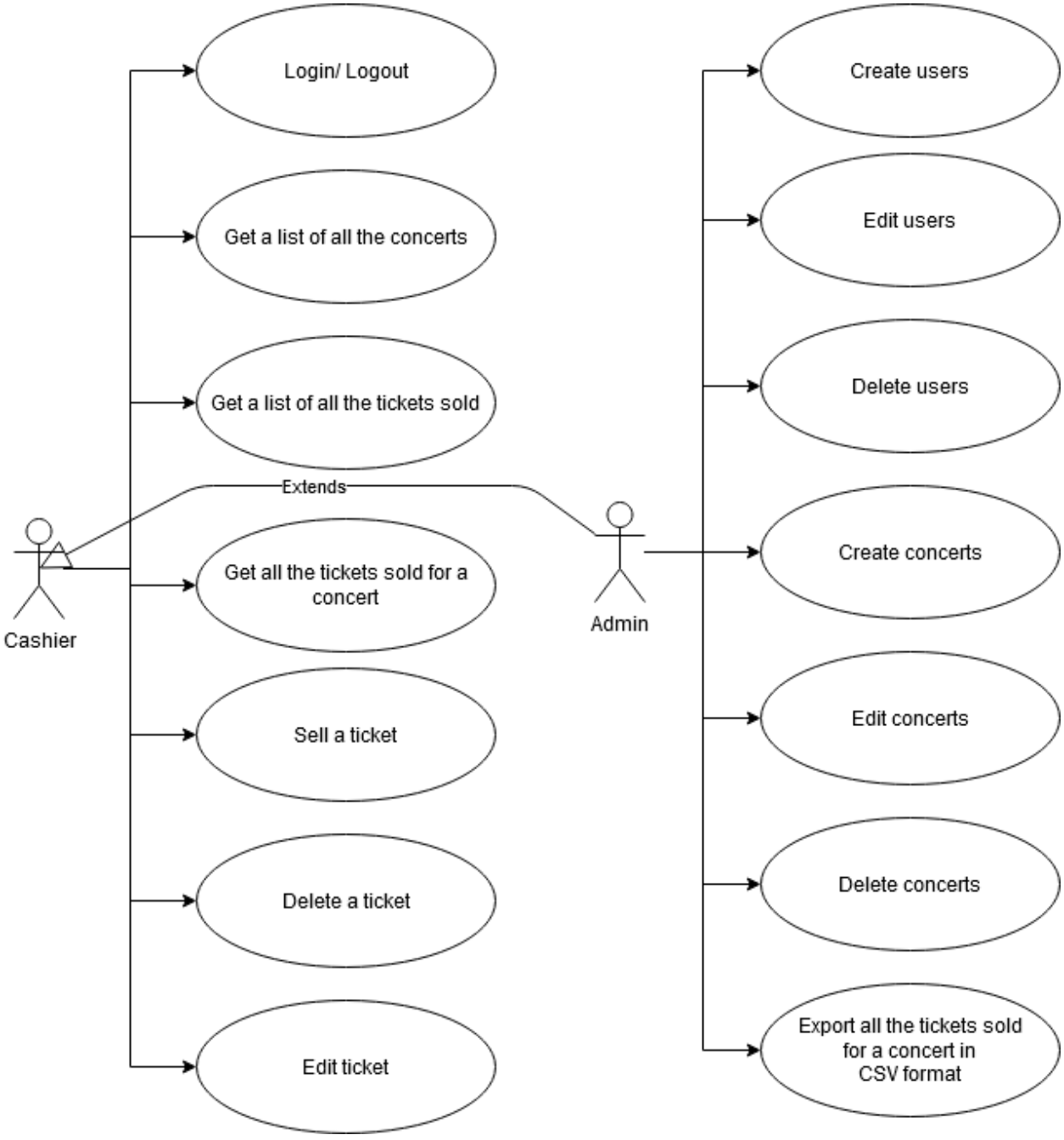
- Provide access to its endpoints using the common HTTP request methods (GET, POST, PUT, DELETE) to get data from the database, create new data, update existing data and delete data. In this context data has the following meanings: tickets, users and concerts.
- Prevent cashiers from exceeding the number of available tickets for a concert when selling new tickets or when editing an existing ticket. If the number of tickets available for sale is smaller than the number of tickets the cashier requests from the API, he will get notified with a HTTP status code 406 (Not Acceptable).
- Multiple user support with different roles for each user. Every endpoint is secured with spring security and only authenticated and authorized users can access them.
- CRUD operations on users, tickets and concerts.
- An excel containing all the tickets sold for a given concert can be generated by the administrator.

1.3 Non-functional Requirements

The non-functional requirements this RESTful API system has are the following:

- **Security**: The system has protection against SQL injection. This is done using parameterized SQL queries. The user passwords are stored in the database using one-way encryption. The algorithm used for encryption is BCRYPT.
- **Availability**: The expected up-time for this system is 99.95%. This translates in a maximum downtime of roughly 22 minutes per month.
- **Performance**: Can work fine under medium load and expected response time is < 100ms.
- **Testability**: The authorization and the ticket selling limit was tested using unit tests. The system is easily testable.
- **Usability**: This API is easy to use because each endpoint has a specific HTTP request method and an easy-to-use data transfer object model in the form of a JSON file.

2. Use-Case Model



Use case: Sell a ticket
Level: user-goal level
Primary actor: Cashier
Main success scenario: The cashier enters all the information for the user correctly and the number of tickets available for sale for the requested concert is smaller or equal than the number of places requested on the ticket.
Extensions: The concert is sold-out or the ticket requests more places than there are available. The cashier gets notified that the transaction cannot be executed.

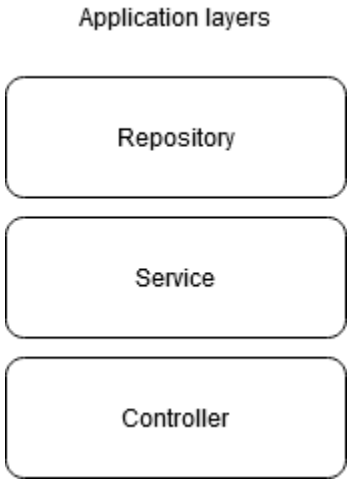
3. System Architectural Design

3.1 Architectural Pattern Description

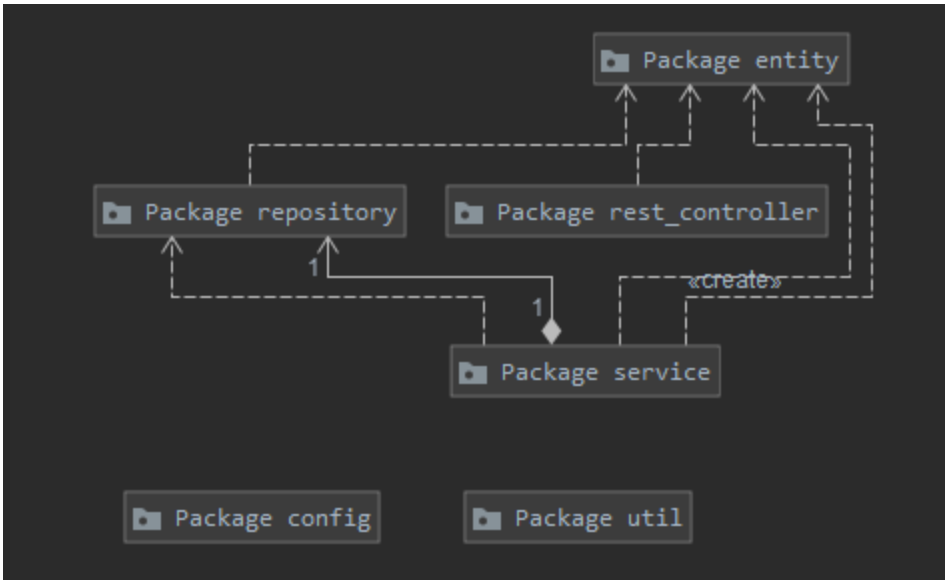
The application uses three-layer architectural design pattern. It is divided in three main layers: Repository, Service and Controller.

3.2 Diagrams

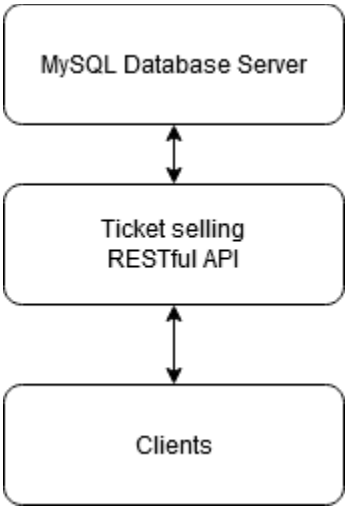
The repository layer is responsible for persisting data to the database and getting the data from the database. The service layer uses the repository layer and is responsible for data validation and supplies data to the controller. The controller layer has the endpoints and uses the service layer to get and send data. Through the controller the application communicates with the external world.



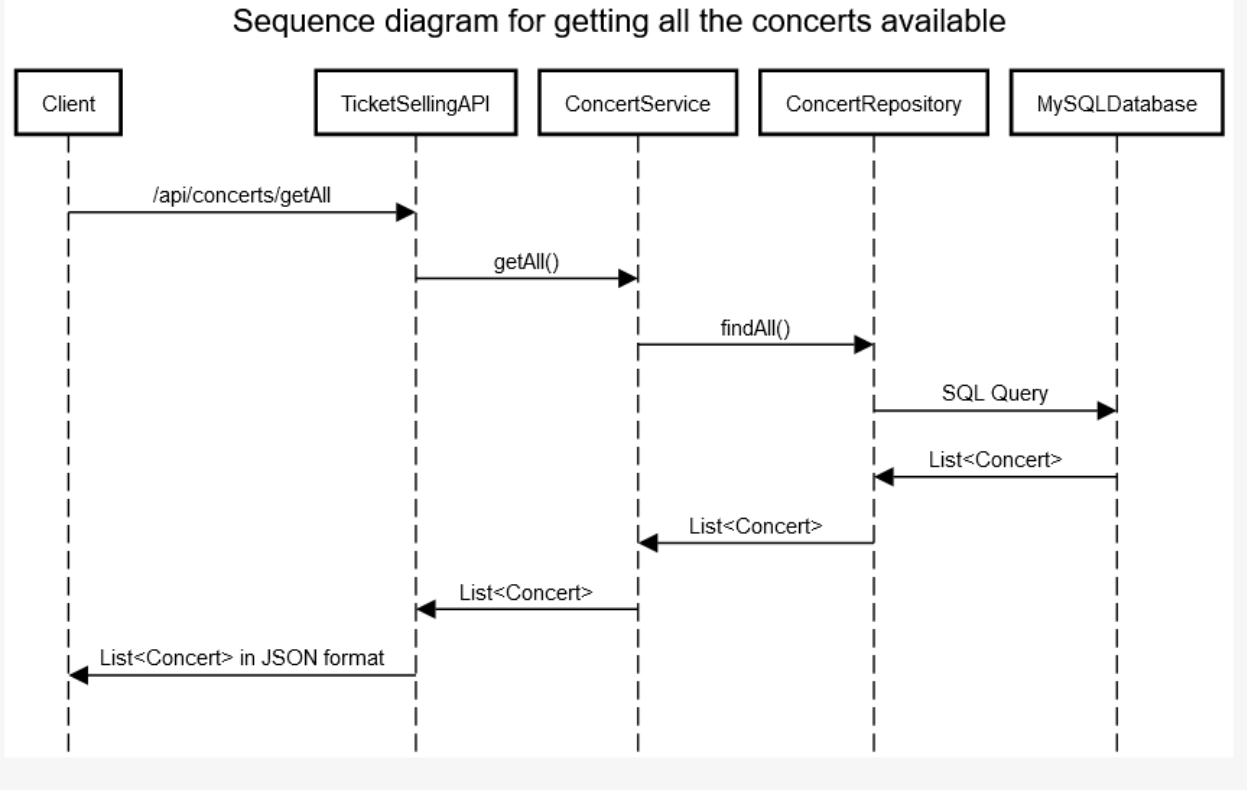
Package diagram:



Deployment diagram:



4. UML Sequence Diagrams

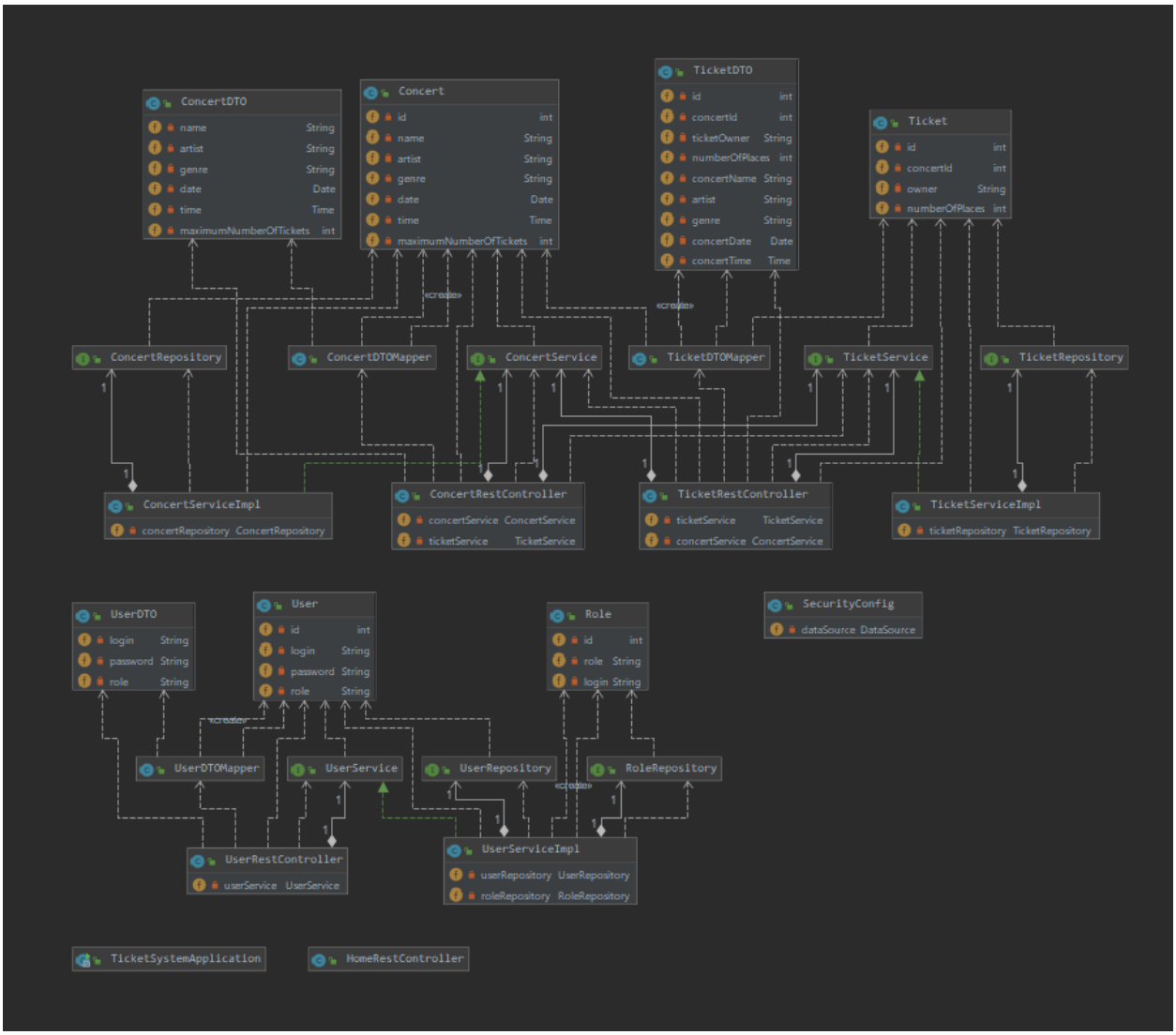


5. Class Design

5.1 Design Patterns Description

The application uses the Factory Method design pattern for csv ticket details document generation. The design was chosen because in the future more document types can be supported for exporting ticket data. For example, PDF documents can be supported.

5.2 UML Class Diagram



6. Data Model

Data models used in the application are:

- Lists – for storing objects
- Map – for initial data generation

7. System Testing

Unit tests were used to test two of the most important use cases of the application. Security was tested first by accessing an endpoint with the wrong authorization and the HTTP Forbidden status was sent. After that the test system tries to access the same endpoint with the correct credentials and authorization and the OK status was received. The ticket selling limit was tested also by generating a test concert and selling tickets successive until the limit was reached and the tickets could no longer be sold.

8. Bibliography

The following sites were used to generate diagrams.
https://sequencediagram.org/
https://app.diagrams.net/