Film Rating Application

Analysis and Design Document

Student: Horvath Andrea - Anett

**Group: 30434**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 04/Apr/21 | 1.0 | First version | Horvath Andrea Anett |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

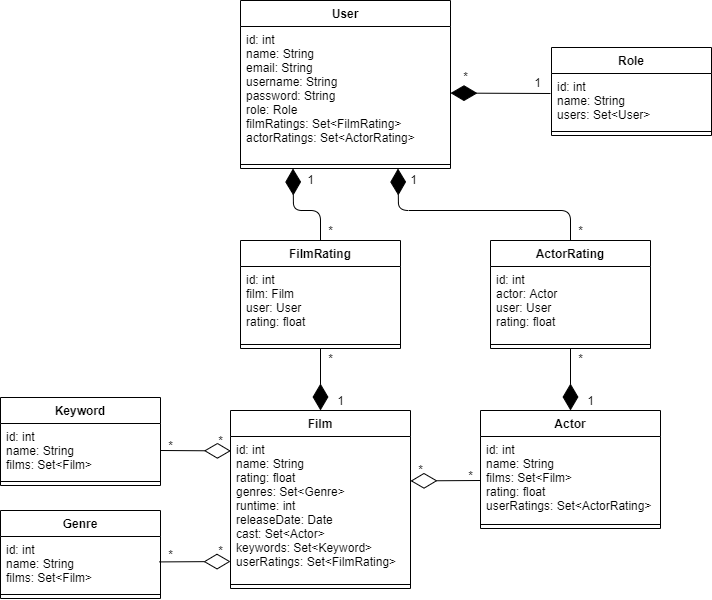
VI. Bibliography 5

# Project Specification

# The Film Rating Application (FRA) will be a web application developed as a client-server system. The FRA will provide film related information, i.e. rating, genres, runtime, director, writer, cast, plot keywords, release date. The possibility for users to rate will be provided in order to have a clear image of the public opinion.

# Elaboration – Iteration 1.1

# Domain Model



# Architectural Design

## Conceptual Architecture

The application will be developed using a layered architecture. Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data (e.g., calculate values or aggregate data), and pass that information up to the presentation layer.



## Package Design

**

## Component and Deployment Diagrams

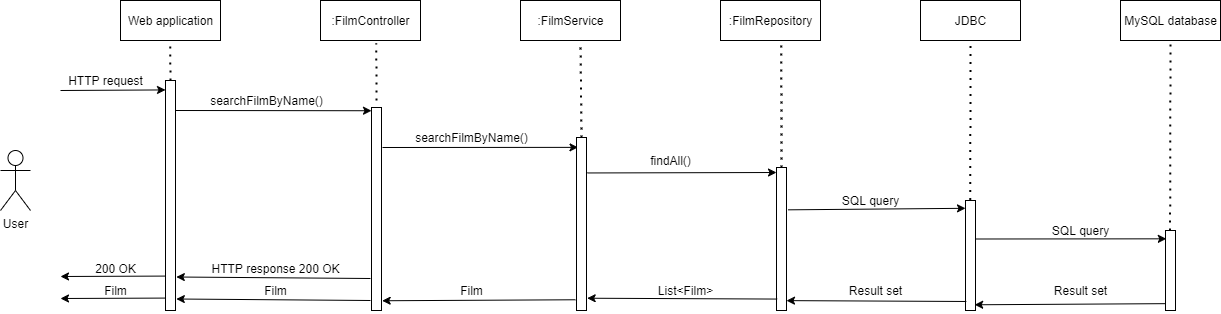
# C:\Users\Andrea\Documents\An3-sem2\SD\LAB\Project documentation\deployment.png

# Elaboration – Iteration 1.2

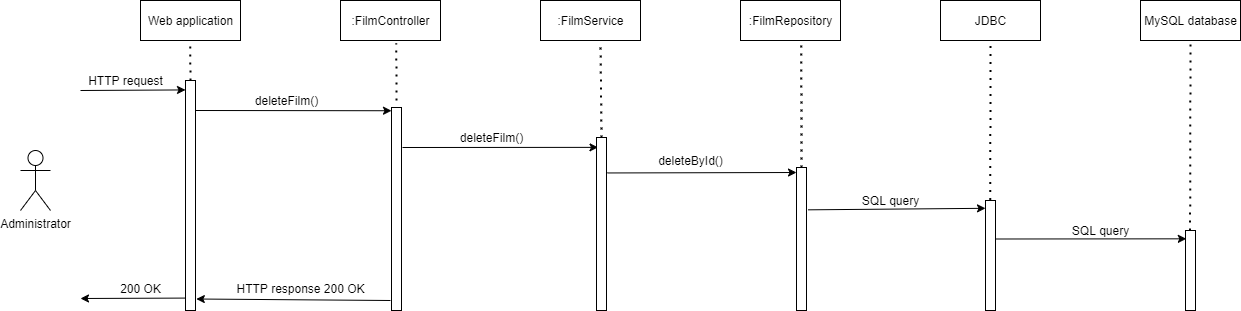
# Design Model

## Dynamic Behavior

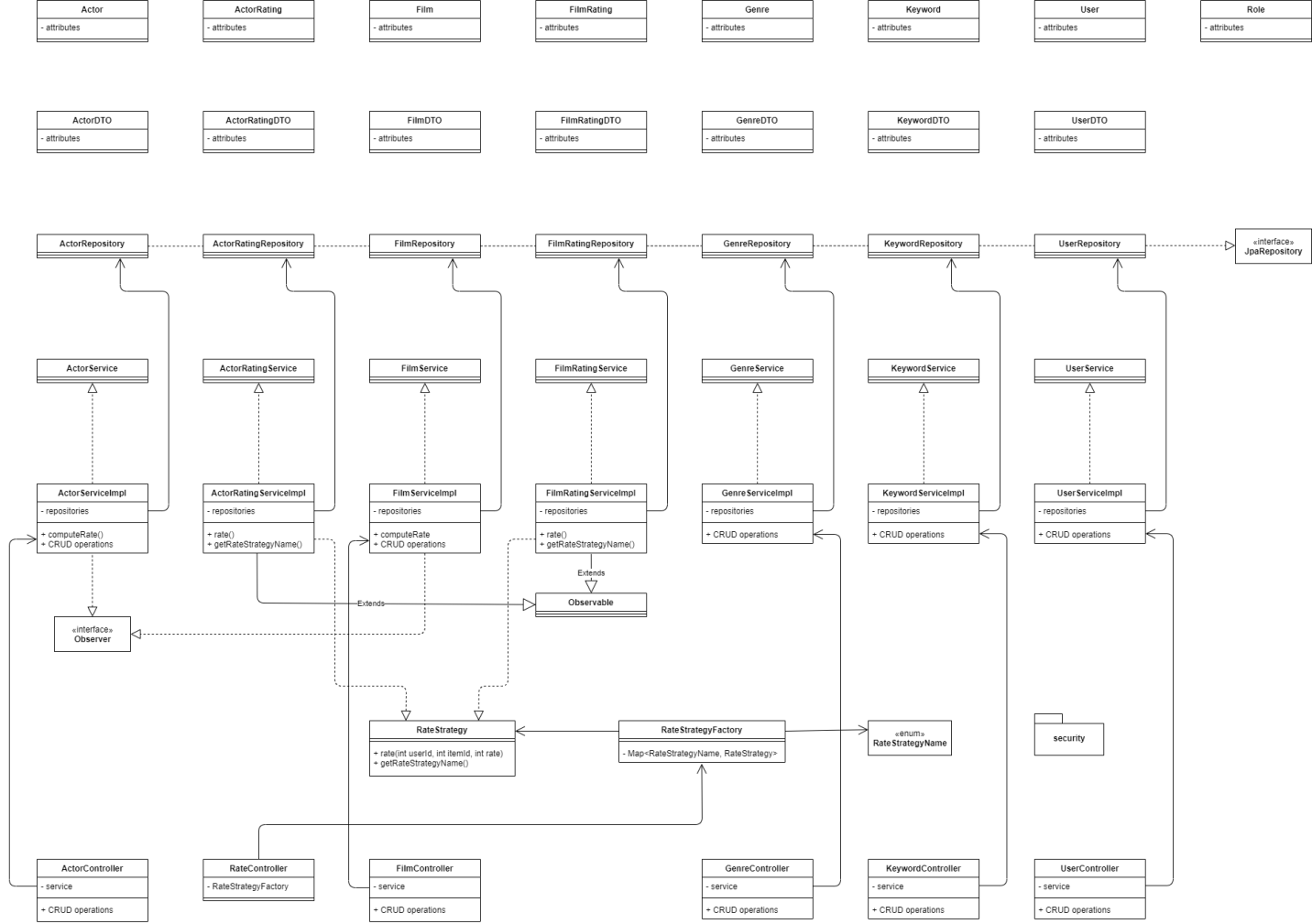
Search film by name



Delete film by id



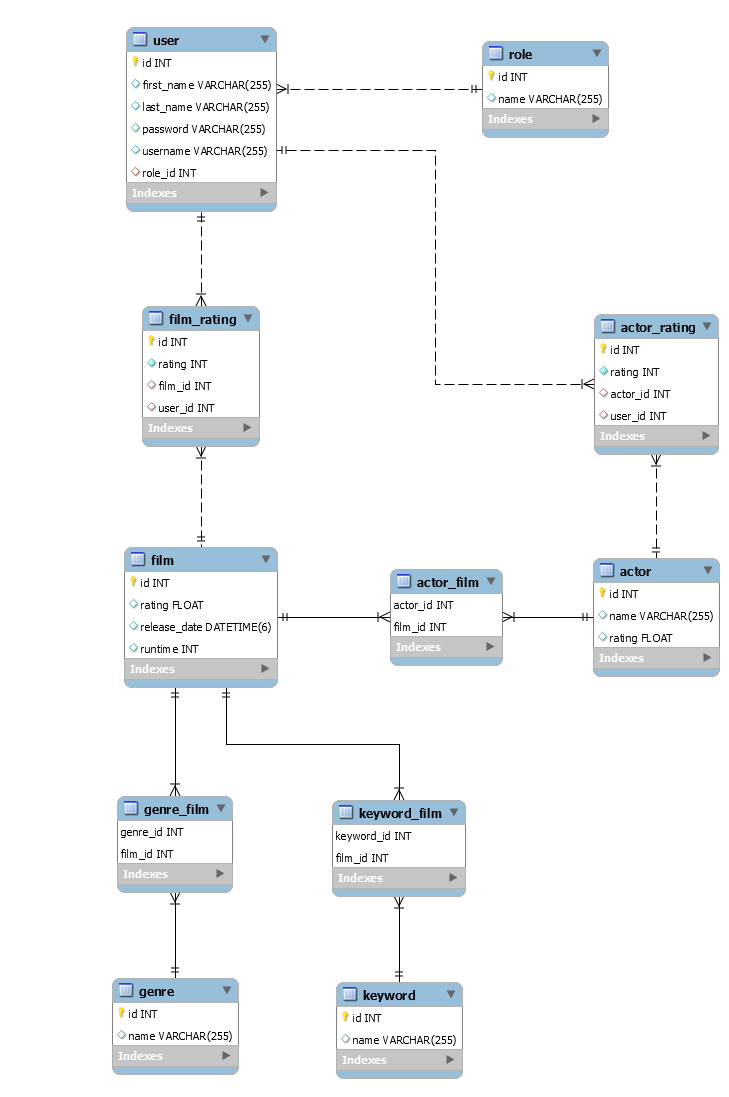
## Class Design



Used design patterns:

* Observer – to notify when a new rating is done in order to log the data and to send a notification email (for both films and actors).
* Strategy – for rating (either actor or film)
* Factory – to create the proper RateStrategy (for actor or for film).

# Data Model



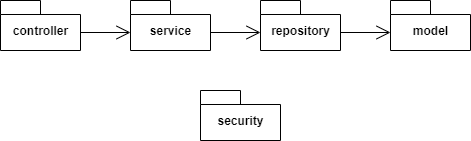
# Unit Testing

In order to test the correctness of the implementation, manual testing using Postman will be used, by creating HTTP requests. For each requested operation, all the scenarios will be tested, both successful and failing scenarios, resulting in Response Entities with suitable messages provided, along with a corresponding HTTP status, depending on the response (OK, CONFLICT, NOT\_FOUND, etc.).

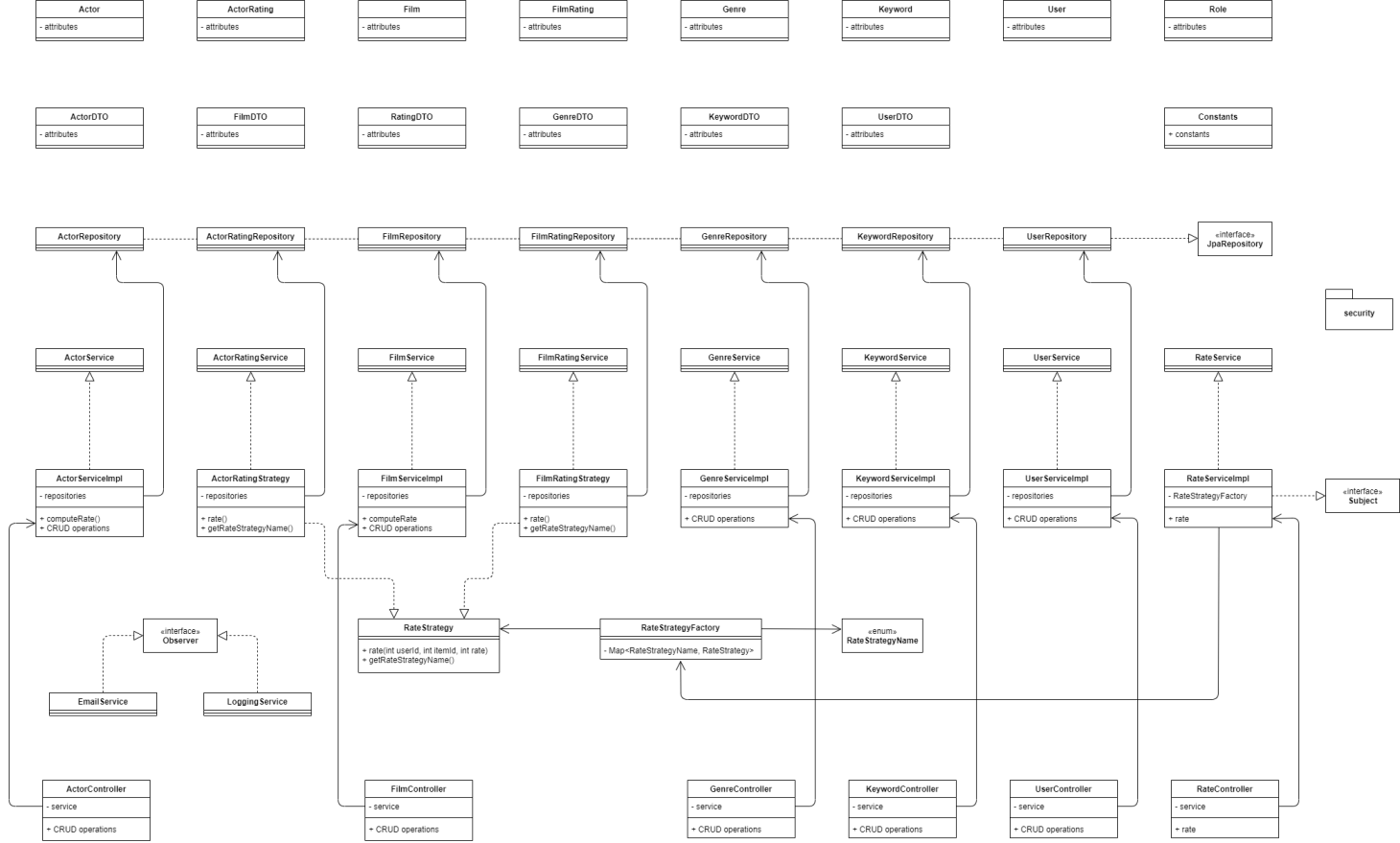
# Elaboration – Iteration 2

# Architectural Design Refinement

A new package was added, i.e. the security package. The new package diagram is the following.



# Design Model Refinement

**

# Construction and Transition

# System Testing

In order to test the correctness of the implementation, manual testing using Postman will be used, by creating HTTP requests. For each requested operation, all the scenarios will be tested, both successful and failing scenarios, resulting in Response Entities with suitable messages provided, along with a corresponding HTTP status, depending on the response (OK, CONFLICT, NOT\_FOUND, etc.).

Methods in *UserService* will be tested using JUnit . We will test the methods *retrieveUserByName* and *retrieveUserByEmail*. For both of them the following cases were tested:

* A success case in which the expected data is found;
* A failure case in which the expected data is null, no user is found.

# Future improvements

As future improvements we can consider:

* Developing an interface
* Extend the notification system; administrators can be notified when a new user is registered

# Bibliography

1. <https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>
2. <https://www.codejava.net/frameworks/spring-boot/email-sending-tutorial>
3. <https://springframework.guru/gang-of-four-design-patterns/observer-pattern/>
4. <https://ravthiru.medium.com/strategy-design-pattern-with-in-spring-boot-application-2ff5a7486cd8>