|  |
|---|
| Programme **Applied Computer Science** |
| Course **Object Oriented Architectures & Secure Development** |
| Lecturers **A. Debou, M. De Wael, F. Vlummens** |

## SOURCE FILES

All required source files (this assignment, specific Java classes, base FXML, SQL) can be found on Leho under the header **"MOCK EXAM"**.

## IMPORTANT

Every form on (on-line) communication between students and other parties is strictly forbidden.

This is an individual assignment. Should an irregularity (such as phone usage, cheating, copying, hacking, use of social media clients, ...) occur, this will lead to a notification of both the student(s) involved and the chair of the examination board, as defined in the Education and Examination Regulations (EER).

After completing this exam, it is forbidden to publish this assignment or the solution using any means possible, except for in your individual repository on the Gitlab server, which has been created for that purpose.

Read the assignment carefully and completely before beginning the exam.

## ALLOWED SOURCES

You may use your own class notes, slides, books, syllabi and other materials, including the internet. Communication with fellow students or third parties is strictly forbidden (see above). Make sure that any applications that might pop up or launch automatically or may be interpreted as an attempt of fraud (Outlook, Messenger, Facebook, ...) are closed!

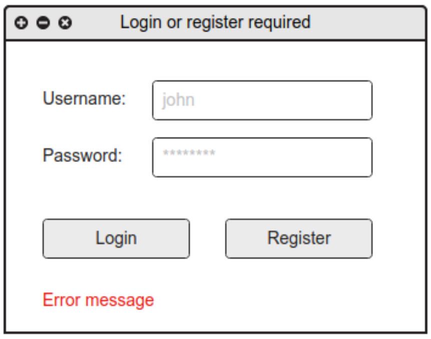
## ASSIGNMENT

### 1. General

Write a multi-tier *JavaFX* application with *gradle* as build tool. The application consists of a login screen and a screen in which users can look up and consult movies through client-server technology. They can also add reviews for the movies they found. These reviews are stored in a local MySQL database.

### 2. Flow description

Upon application launch, users must register/login using the appropriate fields and buttons on the login screen:



Login or register required

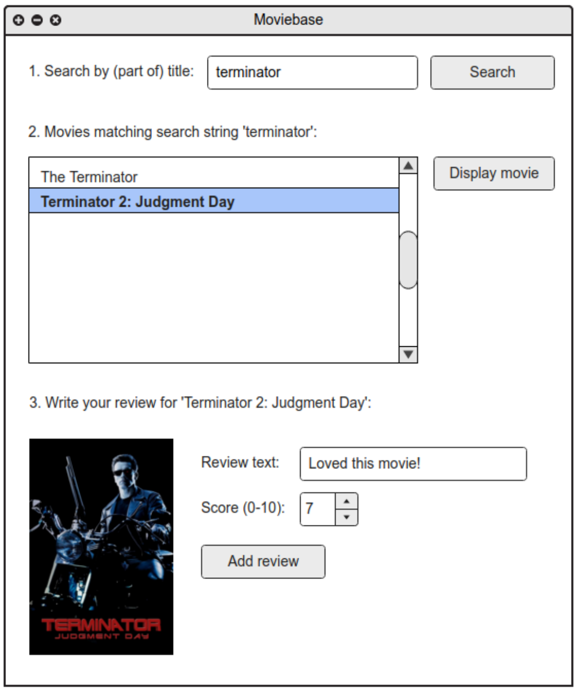
Username: john

Password: ********

Error message

Both actions are performed based on a *username* and *password*, followed by pressing one of the two buttons. Should anything go wrong upon login (e.g. invalid username and/or password) or registration (e.g. username already in use), make sure to display an error message on the screen itself.

Upon successful login or registration, a user is presented with the next screen, in which one can search for movies based on (part of) the title and a click on the **Search** button.



Moviebase

1. Search by (part of) title: terminator

2. Movies matching search string 'terminator':

The Terminator
Terminator 2: Judgment Day

3. Write your review for 'Terminator 2: Judgment Day':

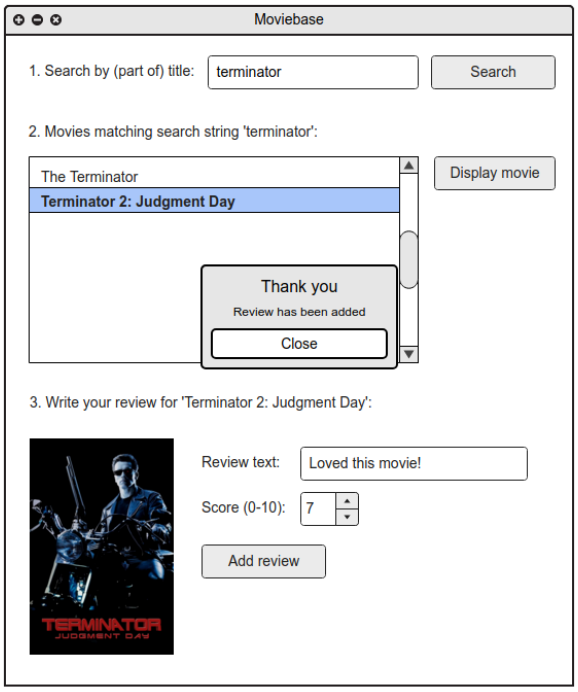
Review text: Loved this movie!

Score (0-10): 7

Any matching movies (hits) are displayed in the list. If the user chooses a movie and clicks the **Display movie** button, the movie cover image is displayed at the bottom of the screen.

Next up, the user needs to add a short text and a score (0 to 10). When clicking **Add review**, this review is added to the local database (see later).

Should anything go wrong when adding the review, show an error message in a dialog. Also, after successful submission of the review, show a confirmation message using a dialog.


Moviebase
1. Search by (part of) title:
terminator
Search
2. Movies matching search string 'terminator':
The Terminator
Terminator 2: Judgment Day
Display movie
Thank you
Review has been added
Close
3. Write your review for 'Terminator 2: Judgment Day':
Review text:
Loved this movie!
Score (0-10):
7
Add review

## 3. Data source summary

- The **user** data (for login/registration) is to be stored by you in the MySQL-database (see §5 regarding the database).
- The general information regarding the **movies** and their lookup is done through the network (see §4 regarding client-server communication).
- The **reviews** the user adds are once again stored in the local MySQL-database (see §5 regarding the database).

## 4. Communication between client and server

The server has already been developed by us. Your multi-tier Java application functions as client and communicates with the server using the technologies we studied. Communication goes as follows:

1. The client connects to port **32768** on the address **that will be provided by the teachers**
2. The client sends an instance of **MovieSearchMessage** over the network, with **query** the search string (e.g. **terminator**).
3. The server responds with one of the following:
    a. An instance of **MovieResultMessage**, whose property **results** provides you with an ArrayList of **Movie** objects matching the query.
    b. An instance of **ErrorMessage** should something go wrong. The property **message** further details what went wrong.

To make sure client and server speak the same language, they need the same versions of the message classes **MovieSearchMessage**, **MovieResultMessage** and **ErrorMessage** (all of which inherit from **Message**), as well as the domain class **Movie**. You get these five classes from us (see Leho) in the ZIP file **classes.zip**.

To ensure messages and movies can be transmitted over the network, **you will need to add something at the right place(s) of the code.**

Also place the provided classes in the correct/relevant packages.

## 5. Database

- The database contains two tables: one with users (username and password) and one with reviews (review id, username, movie id, review text and score). Make sure your Java application follows the OO principles.

  Use the SQL script **moviebase.sql** to build the database. Besides that, we also provide you with the SQL statements to add users and reviews. These can be found in **statements.txt**. Both files are on Leho.

- Ensure database access is as correct/safe as possible. Of course, we want to avoid somebody from executing things on the database server that are not necessary for the application. Also add a screenshot showing how you did this. Place this screenshot with the name **securedb.png** in the root of your repository.

  It should be possible to connect to a different MySQL server or database or even with different credentials, without modifying the Java source code itself. Make sure these connection details cannot easily fall in the wrong hands.

  The username for the database should be **moviebase-user** and associated password **moviebase-pwd**.

## 6. Non-functional requirements

- Make sure clean and readable code (and configuration) is used throughout the entire solution, according to the best practices.
- Apply Oracle's secure coding guidelines where necessary.
- Use logging.
- Throughout your code, use a self-made exception called MovieException. Provide an appropriate message.
- Provide one relevant JUnit5-test, created according to the structure we studied.