Project Documentation

ISTE-330 Database Connectivity and Access

FilmFinder

Document (filename): “ISTE-330 - Team 7 - \_FilmFinder\_ Project Documentation.docx”

Author(s): Filipa Ivanković, Lovro Mavrlja, Fran Barisic, Karlo Bozikov, Vid Kraljic

Date (last change): Apr 28, 2023

Version (last version): 5.3

| *Version* | *Description of Change* | *Author(s)* | *Date* |
| --- | --- | --- | --- |
| 0.1. | Project begins - initial idea, overview, team organization, coordination and communication | All members | 2023-03-20 |
| 0.2. | Documentation - M0 - Chapter 1: introduction, overview, project scope, background, references and document overview | All members | 2023-03-21 |
| 0.3. | Documentation - M1 - Architecture, ERD diagram & explanation, Details about the layers, Expanding from chapter 1 | All members | 2023-03-26 |
| 0.4 | Documentation - M1 - Partial textual explanation of the ERD, Expanded “Project Scope”, “Document Overview” written-up | All members | 2023-04-03 |
| 1.0 | Database and database data completed | All members | 2023-04-03 |
| 2.0 | Data Access Layer - Logic for accessing values and data from MySQL implemented for Actors, Movies and Genres tables. Each CRUD operation is functional and working. Tables can be fetched, deleted, updated and inserted into.  Database - Connection with MySQL database and tables established. Communication working without interruption.  Business Layer - DataManipulator class takes the data provided by the models and transforms it into objects of those models. Currently available for Movies, Actors and Genres tables. | All members | 2023-04-09 |
| 3.0 | Data Access Layer - Movies\_Genres table started - Retrieve all Genres for Movie working. Retrieving all Movies for a Genre needs to be implemented.  Business Layer - Methods for all CRUD operations implemented  Presentation Layer - Interface, skeleton and design created for the User interface | All members | 2023-04-16 |
| 4.1. | Data Access Layer- Movies\_Genres, Movies\_Directors, Movies\_Actors and Directors tables implemented.  Business Layer - Each model’s values are accessible through the model’s objects and usable by every other part of the application. The Data Manipulator class has access to each model and can call any CRUD and additional functions for each model | All members | 2023-04-19 |
| 4.2. | Presentation Layer - Movies are read and displayed on the UI. Login, Actors, Directors and Movies UI screens are completed. | All members | 2023-04-23 |
| 5.1. | **Data Access Layer** - Added models for User\_Ratings & Reviews for added User + Login functionality.  Genres model - fixed an issue when trying to insert or update a genre that already exists (genre names must be unique)  Movie\_Actors, Movie\_Directors, Movie\_Genres - Improved logic of code to utilize MySQL” JOIN” statements. Functionality remains the same, but less DRY violation and less wordy.  **Business Layer** - DataManipulator - Added methods & logic for User\_Ratings & Reviews models | All members | 2023-04-26 |
| 5.2. | Data Access Layer - model for Movie\_Collections, Movie\_Collections\_Items, Watch\_History  Business Layer - Data Manipulator - logic for aforementioned models  Database - Fixing some issues, and adding some more data for testing. | All members | 2023-04-27 |
| 5.3. | Presentation Layer fixes  Bug fixes  Final documentation | All members | 2023-04-28 |

Table of contents

[1 Introduction 5](#_30j0zll)

[1.1 Overview 5](#_1fob9te)

[1.2 Purpose and Scope 5](#_3znysh7)

[1.3 Background 5](#_2et92p0)

[1.4 References 6](#_tyjcwt)

[1.5 Document Overview 6](#_3dy6vkm)

[2 Problem Description and Solution Architecture 8](#_1t3h5sf)

[2.1 Problem Description 8](#_4d34og8)

[2.2 Technologies and Architectural Design 8](#_2s8eyo1)

[2.3 Database Layer and Database Connectivity Layer 9](#_17dp8vu)

[Tables structure: 11](#_oam18picdv1s)

[2.4 Business Layer 15](#_3rdcrjn)

[2.5 Presentation Layer 15](#_26in1rg)

[2.6 Areas of particular concern 16](#_lnxbz9)

[3 Requirements 18](#_35nkun2)

[3.1 Context 18](#_1ksv4uv)

[3.2 Functional Requirements 18](#_716yywd6o84)

[3.3 Other (Non-Functional) Requirements 19](#_2jxsxqh)

[4 User Documentation 21](#_z337ya)

[4.1 Graphical User Interface Design 21](#_3j2qqm3)

[4.2 User Manual 23](#_1y810tw)

[5 Installation, Configuration and Acceptance Testing 24](#_4i7ojhp)

[5.1 Installation 24](#_2xcytpi)

[5.2 Configuration 24](#_1ci93xb)

[5.3 Acceptance Testing 25](#_3whwml4)

[6 Final Remarks and Conclusion 26](#_2bn6wsx)

# Introduction

The “FilmFinder” application will be a Java and MySQL-based application that aims to provide users the ability to search and browse through a collection of movies & series, and view their details (such as genre, director, actors, synopsis, images for the movie/series etc.). The application should also allow users an interactive option of being able to contribute to the existing database by adding, editing and deleting movies and TV series.

## Overview

FilmFinder is a straightforward online movie and series library application that enables users to explore and find a wide variety of films and TV series. The main goal of this project is to make discovering movies and series easy and enjoyable by offering helpful search and filter options. With a user-friendly interface, FilmFinder allows users to browse through an extensive catalog of movies and series, as well as create their personal watchlists and collections. The application focuses on helping users find and learn about movies and TV shows without providing any streaming functionality. Overall, FilmFinder aims to offer a simple and engaging experience for movie and series fans who are looking for an easy-to-use platform to discover new content.

## Purpose and Scope

This document serves as a comprehensive guide to the FilmFinder project, detailing its objectives, features, and underlying structure. The purpose of this project is to develop a user-friendly online platform for discovering movies and TV series, catering to a diverse audience of film and series enthusiasts. The scope of this document encompasses the high-level description of the application, its components, and the technology stack used for development, while not delving into detailed implementation specifics. The intended audience for this document includes project managers, developers, and clients who are interested in understanding the overall concept, goals, and structure of FilmFinder. By providing a clear overview of the project, this document aims to facilitate effective communication and collaboration among all stakeholders involved in the development process.

This document will provide a thorough and detailed explanation of the FilmFinder application. This will include the general outline of the project, the architecture and structure, functionalities of the application, explanation of the multi-tier architecture present in the application, and details on how to set up and install the necessary components to run the application. This document will not go into detail on how to use the application, as one of the goals of the application is to make it simple enough so that even someone who may not be familiar with databases and computer applications may be able to use and understand it with some trial and error.

## Background

Team 7 is a group of five dedicated students, each with a minimum of four years of academic experience: Fran Barisic, Karlo Bozikov, Filipa Ivanković, Vid Kraljic, and Lovro Mavrlja. They are proficient in Java programming and MySQL database management. Working together, the team aims to deliver a high-quality movie and series database Java application within the project's designated time frame. They will utilize their collective knowledge and skills to create an easy-to-use application that enables users to efficiently find information about their favorite movies and series.

Filipa Ivanković will serve as the team's project coordinator, overseeing the submission of documentation and project materials on MyCourses. She will also act as the liaison with the sponsor professor, Branko Mihaljević, keeping him informed about the team's progress and addressing any potential issues or challenges that may arise during the development process.

## References

FilmFinder is an application primarily built using Java as the programming language and MySQL for database management. The Java programming language handles the application's logic and communication with the MySQL database, which stores all movie and series-related information, such as title, synopsis, image path, year of release, directors, actors, and user reviews. The JDBC driver establishes a connection to the database and is responsible for executing SQL queries.

Below is a list of essential links and resources related to the technologies and tools used in the development of FilmFinder:

* Java programming language (JDK 17) - Download Java: https://www.java.com/en/download/
* MySQL database - Download MySQL: https://www.mysql.com/downloads/
* JDBC driver for MySQL(ver. j-8.0.32.jar) - Download MySQL Connector/J: https://dev.mysql.com/downloads/connector/j/
* MySQL Workbench(8.0CE) - Download MySQL Workbench: https://dev.mysql.com/downloads/workbench/
* GitLab (code versioning): https://about.gitlab.com/
* JavaFX (for desktop application - 20 ) - JavaFX: https://openjfx.io/
* IntelliJ IDEA (Java IDE - 2021.3.1. ) - Download IntelliJ IDEA: https://www.jetbrains.com/idea/download/
* Visual Studio Code - Download Visual Studio Code: https://code.visualstudio.com/Download

These resources provide essential information and downloads for the various technologies and tools employed in the development process of FilmFinder, ensuring that the team has access to the required software and guidance.

## Document Overview

The first and titular page of the document provides the list of changes done throughout the project’s development. Each change will have a short description, members who worked on that change, version, and the date when the changes were made. The next page contains the table of contents on which the reader can easily see the outline of the whole document.

The “Chapter 1” section of the document will focus on the general description of the FilmFinder application, as well as introduce the team working on the project, and references to software that has been used for the development of the project. Chapter “1.0” is the introduction which shortly describes the purpose and objective of the FilmFinder app. Chapter “1.1.” and “1.2.”, the “Overview” and “Project Scope”, are the continuation of the Introduction, and go more in depth about what the goal and capabilities are of the FilmFinder application. Chapter “1.3.” is a short introduction of the team working on the project. Lastly, “References” lists and explains the software that was used to create the FilmFinder application. Links to those softwares are provided as well.

Chapter 1 explains the “what” of the FilmFinder project, and Chapter 2 answers the questions of “Why?” and “How?” the FilmFinder application operates. This chapter explains how the FilmFinder’s multi-tier architecture is constructed. The “Problem Description” chapter explains the concept and idea behind the FilmFinder application and why the application was made in the first place. Also, this chapter compares some of the FilmFinder’s functionalities with similar competitors such as “Rotten Tomatoes” and “IMDB”. An “entity-relationship diagram” is provided which is both described through text and a visual image that outlines the whole database structure. The multi-tier architecture is described in detail, with each of its layers described and explained: “Business layer”, “Database layer”, “Presentation layer” and “Data Access Layer”. The last part of Chapter 2 lists out the possible “Areas of particular concern” where potential problems or concerns that may be encountered throughout the project’s development.

Chapter 3 is all about FilmFinder’s functionalities. This chapter outlines and shortly explains each of the application’s possible functionalities that should be working 99% of the time, baring some possible unforeseen issues or circumstances.

Chapter 4 is “User documentation”. This chapter will provide some instructions for the user on how to use some of the FilmFinder’s core functionalities. Information about what the user can expect, what to do in case of errors or confusion, and general information on how to navigate through the FilmFinder application with as little issue as possible.

Chapter 5 will focus on the setup process and installation of the FilmFinder application. Information on how to install and configure the application will be provided and described here.

Lastly, and finally, Chapter 6 consists only of the “Final Remarks and Conclusion”. This chapter will be filled out near the end of the project’s development and will include potential concerns, expectations and overall conclusion to the FilmFinder development. This chapter will also outline all the possible deliverables our team will be required to provide for the FilmFinder application.

# Problem Description and Solution Architecture

**Problem Description**:

The project aims to provide an easy-to-use platform for users to explore and find information about movies and TV series efficiently.

**Solution Architecture**:

FilmFinder will use a multi-tier architecture, consisting of the following layers:

a) Presentation Layer: Handles the user interface, employing JavaFX for desktop or HTML, CSS, and JavaScript for web-based UI.

b) Business Logic Layer: Implements core functionality using Java for processing user requests and managing communication with the database.

c) Data Access Layer: Manages the connection between the application and the MySQL database using the JDBC driver.

d) Database Layer: Stores all movie and series-related data in a MySQL database.

This architecture ensures an organized and efficient design for the FilmFinder application.

## Problem Description

The “FilmFinder” database application aims to provide a user-friendly platform for searching and browsing through a vast collection of movies and TV series. The primary objective is to provide a reliable and easy to use desktop application which will allow users to access valuable information about their favorite movies and series.

Currently, there exist several solutions that provide movie and TV show information, such as IMDB and Rotten Tomatoes. However, these solutions are limited in terms of user customization and accessibility. Users may not be able to search and filter their results on these websites or get specific information about a particular movie or TV show. These websites serve more so as a quick way of seeing user and critics ratings on new and upcoming movies.

The proposed solution will leverage Java and MySQL to provide a more comprehensive and accessible solution. Users will have the ability to filter search results based on various criteria such as title, genre, year of release, rating, and other relevant criteria. Additionally, the database will store detailed information about each movie or TV show, including plot summaries, cast and crew details, trailers, and user ratings.

If time allows, the proposed solution will also provide an interactive platform for users to add, edit, and delete movies and TV shows. This feature will allow users to contribute to the database and keep it up-to-date with the latest information.

## Technologies and Architectural Design

FilmFinder's architectural design is based on important software design decisions aimed at creating an efficient and scalable application. The choice of technologies and the multi-tier architecture ensures modularity, maintainability, and ease of development.

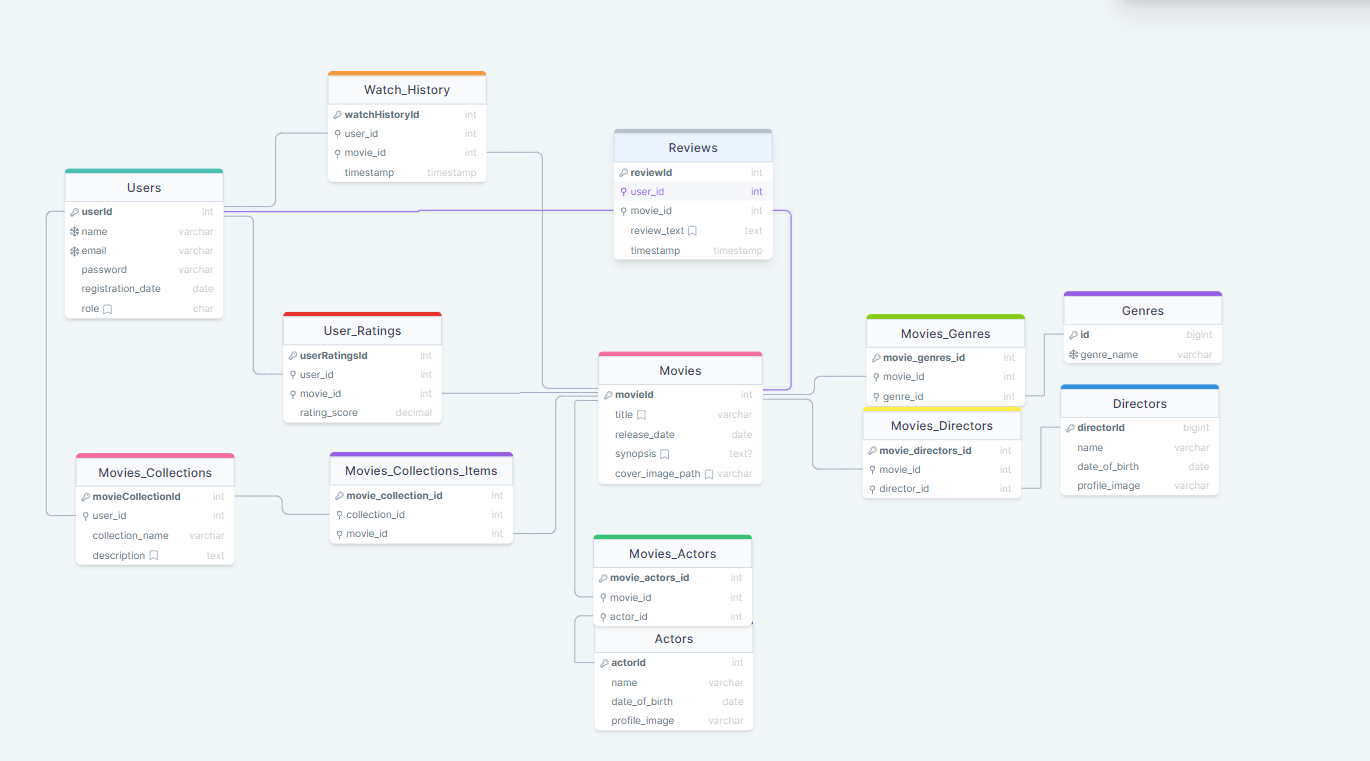
Programming Language: Java was chosen for its versatility, wide adoption, and extensive library support. Java's object-oriented nature allows for clean and modular code, making it easier to maintain and expand the application in the future.

Layers and Rationale: The multi-tier architecture consists of four layers - Presentation, Business Logic, Data Access, and Database. This design promotes separation of concerns, allowing for independent development and testing of each layer. It also enhances scalability and maintainability.

Technologies and architecture:

Presentation Layer: JavaFX for desktop or HTML, CSS, and JavaScript for web-based UI.  
Business Logic Layer: Java for processing user requests and managing communication with the database.  
Data Access Layer: JDBC driver for connecting to the MySQL database.  
Database Layer: MySQL for storing movie and series-related data.

## Database Layer and Database Connectivity Layer



**Movies table**

The primary and center piece table of the database architecture is the “Movies” table. On the diagram it can be seen that the “Movies” table is connected to nearly every other table in the architecture. The “Movies” table will hold the data for the movies such as the movies’ title, release dates, synopsis and “cover image path”. The image poster for each movie will not be stored on the database as this might overload the database and significantly lower its performance; therefore, only a textual string, that is, part of the path to the image. The image that will be stored locally on the application itself.

**Actors table**

The “Actors” table will hold the names, date of birth and the image path for each actor that is present within the movies present in the database. As it is the case with the “Movies” table, the image path is only stored for the image. It is connected to the Movie\_Actors table, which connects the Actor to the Movies they have played a part in.

**Movies\_Actors**

This table will be the connecting table between the “Movies” and “Actors” table. This table will hold the foreign keys for the “movie\_id” and the “actor\_id”, which will be the Primary keys for this table.

**Directors table**

Hold the data for the directors. Same as the Actors table, only a part of the image path is present in the MySQL Db itself. Connected to the Movie\_Directors table, which connects it to their corresponding movies.

**Movie\_Directors table**

Connecting table between Movies and Directors table. The movies and directors id are the foreign keys of their corresponding tables.

**Genres table**

Table which holds the names of the genres. Genre names must be unique, and therefore, one cannot insert or update genre names to existing names.

**Movies\_Genres table**

Connecting table between Movies and Genres tables. Holds the foreign keys for the primary keys of the Genres and Movies tables

**Users table**

Table which holds the values for all registered users in the DB. Users have unique usernames and emails. Users will have to use their username and password to connect to the DB; however, it is possible to use the application without the need for logging in. Logging in only provides some additional features and functionalities. The “role” row represents the level of permissions a user has. A user by default is a “u” type of user. Admins are given the “a” role, and they have a wider ability of privileges and accessibilities. Guests are “g”.

**Watch\_History**

Table which holds a certain users viewed movies. Each movie is accompanied by a timestamp for when the movie has been placed into the watch history. Holds the foreign keys for the primary keys of the Movies and Users tables.

**User\_Ratings**

Table which holds the movie scores/ratings. Foreign keys for the movie for which the score is for, and the foreign key for identifying the user who left the rating.

**Reviews**

Table which holds the user written reviews for a particular movie. Timestamp for when the review has been made. Foreign keys for the user who made the review, and the key for the movie for which the review was made for.

**Movie\_Collections**

A table for user made collections of various movies. Users select which movies to add to their collection.

**Movie\_Collections\_Items**

Connecting table between the Movies and Movie\_Collections table. Points to the primary keys of the movie and the movie\_collection. This table is responsible for pointing a collection toward a movie that has been put there by the user.

### Tables structure:

| **Movies** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| movieId | int | Auto\_Increment Primary Key | 1,2,3 |
| title | Varchar(255) | Name of the movie. Not null, Default(“MovieTitle”) | “The Thing”, “Shrek” |
| release\_date | Date | Year of movie’s release | 2001 |
| cover\_image\_path | Varchar(255) | Path to the image which will be stored on the application. Default(“”) | “The\_Thing.jpg” |

| **Movies\_Actors** | | |
| --- | --- | --- |
| **Column** | **Type** | **Additional Information** |
| movie\_actors\_id | int | Primary Key. Auto\_Increment. |
| movie\_id | int | Foreign key taken from “Movies” table |
| actor\_id | int | Foreign key taken from “Actors” table |

| **Actors** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| actorId | int | Primary Key. Auto\_Increment. | 1 |
| actor\_name | Varchar(255) | First & Last name of the actor | “Sam Worthington” |
| date\_of\_birth | Date |  | 1990-03-01 |
| profile\_image\_path | Varchar(255) | Path to the actor’s image stored locally on the application | “Sam\_Worthington\_jpg” |

| **Genres** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| genreId | int | Primary Key. Auto\_Increment. | 1,2,...14 |
| genre\_name | Varchar(255) | Unique. Name of the genre | “Horror”, “Action”, “Fantasy” |

| **Movies\_Genres** | | |
| --- | --- | --- |
| **Column** | **Type** | **Additional Information** |
| movie\_genres\_id | int | Primary Key. Auto\_Increment. |
| movie\_id | int | Foreign key taken from “Movies” table |
| genre\_id | int | Foreign key taken from “Genres” table |

| **Directors** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| directorId | int | Primary Key. Auto\_Increment. | 1,2,3 |
| director\_name | Varchar(255) | First & last name of director. | “John Carpenter” |
| date\_of\_birth | Date | Birth year, month and day | 1948/01/16 |
| profile\_image\_path | Varchar(255) | Path to the director’s image stored locally on the application | “./img/John\_Carpenter.jpg” |

| **Movies\_Directors** | | |
| --- | --- | --- |
| **Column** | **Type** | **Additional Information** |
| movie\_directors\_id | int | Primary Key. Auto\_Increment. |
| movie\_id | int | Foreign key taken from “Movies” table |
| director\_id | int | Foreign key taken from “Directors” table |

| **Users** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| userId | int | Primary Key. Auto\_Increment. | 1,2,3 |
| username | Varchar(255) | Unique username | “MovieFan123”, “ActionMovies\_Fan” |
| email | Varchar(255) | Unique email | “john.doe@gmail.com” |
| password | Varchar(255) | Should be encrypted | “dsakdajk123” |
| registration\_date | Date | The date for when the account was created | “2023/03/26” |
| role | Char(1) | Either “u” for users, or “a” for admin. Set to “u” by default, and Admin privilege will need to be given manually by another existing admin.  Guest are “g”. | “u”,”g” ”a” |

| **Movies\_Collections** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| movie\_collections\_id | int | Primary Key. Auto\_Increment. | 1 |
| user\_id | int | Foreign key taken from “Users” table | 1 |
| collection\_name | Varchar(255) | Name given to the collection by the user | “Favorite Movies”, “Horror Movie Collection” |
| description | Varchar(255) | Default(“”) | “Movies from the year 2023.” |

| **Movies\_Collections\_Items** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| movie\_collection\_id | int | Primary Key, Auto\_Increment | 1 |
| collection\_id | int | Foreign key taken from “Movies\_Collections” table | 1 |
| movie\_id | int | Primary Key & Foreign key taken from “Movies” table | 1 |

| **User\_Ratings** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| id | int | Primary Key. Auto\_Increment. | 1,2,3 |
| user\_id | int | Primary Key & Foreign key taken from “Users” table | 1,2,3 |
| movie\_id | int | Primary Key & Foreign key taken from “Movies” table | 1,2,3 |
| rating\_score | decimal | Decimal value from 1-10 as a rating for the movie | “5”,”7” |

| **Reviews** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| reviewId | int | Primary Key. Auto\_Increment. | 1 |
| user\_id | int | Primary Key & Foreign key taken from “Users” table | 1 |
| movie\_id | int | Primary Key & Foreign key taken from “Movies” table | 1 |
| review\_text | Text | Default(“”) | “” |
| timestamp | Timestamp | Timestamp of when the review was submitted | 2022/02/01 01:01:22 |

| **Watch\_History** | | | |
| --- | --- | --- | --- |
| **Column** | **Type** | **Additional Information** | **Sample Data** |
| watchHistoryId | int | Primary Key. Auto\_Increment. | 1,2,3 |
| user\_id | int | Primary Key & Foreign key taken from “Users” table | 1,2,3 |
| movie\_id | int | Primary Key & Foreign key taken from “Movies” table | 1,2,3 |
| timestamp | Timestamp | Timestamp for when the watching plan was made for the particular movie | “2023/03/26 20:25:17” |

## Business Layer

The Film Finder's Business Layer focuses on managing the application's core logic and acts as an intermediary between the Presentation Layer and the Database Connectivity Layer. Utilizing Java classes and adopting the Model-View-Controller (MVC) design pattern, the layer efficiently processes user requests, such as searching for movies and managing watchlists.

Within the Business Layer, distinct components like services and controllers handle various aspects of the application, including user authentication, search queries, and data manipulation. This modular approach ensures maintainability and scalability.

To facilitate communication with the Database Connectivity Layer, the Business Layer employs data access objects (DAOs) for executing CRUD operations and data transfer objects (DTOs) for transferring data between layers. Overall, the Business Layer plays a vital role in ensuring smooth communication between the layers while maintaining a clean and well-structured codebase.

## Presentation Layer

*[Provide Presentation Layer description, graphical user interface (GUI) design, including structure, layout and explanations, as well as a description of used technologies. At least several sentences are expected to describe how the Presentation Layer will be constructed, what its purpose is, and how it communicates with Business Layer (below) and the users (clients). You can also include all possible actions, menus, and options. You should start by doing some prototypes or wireframes for* ***deliverables and milestones #1 to #3*** *and finish it no later than* ***deliverable and milestone #4****. In the final* ***deliverable #5*** *you can even provide some screenshots.*

The Presentation Layer of The Film Finder is designed using JavaFX, which offers a modern, intuitive GUI for users to interact with. The GUI is structured with a side menu for easy navigation, a search bar for quick search, and a grid or list view for displaying movie. The purpose of the Presentation Layer is to provide an interactive and visually pleasing interface for users to interact with the Film Finder application. It communicates with the Business Layer to retrieve and display information about movies and user reviews, and to handle user actions such as searches, adding movies to watchlists, and creating custom collections.

The Presentation Layer uses JavaFX for its GUI, which allows for the creation of dynamic and engaging user interfaces. The layout is designed to be clean and intuitive, with easy-to-use navigation and a focus on displaying relevant movie information. The GUI is designed to be responsive, adapting to different screen sizes and resolutions.

Menus and options in the interface include the ability to search for movies, view movie details, add movies to watch lists, and manage user profiles.

Overall, the Presentation Layer provides a seamless user experience for exploring and discovering movies and episodes, with a modern, visually appealing interface and easy-to-use functionality.

## Areas of particular concern

*[In this chapter, you should provide identification of areas of particular note or concerns. It could be about prerequisites (which must be respected) and assumptions, as well as possible risks for your project. Those could be related to an organization, planning, resources, technologies, and availability, as well as team members. You can describe a plan on how to mitigate those risks. You should start doing it for* ***deliverables and milestones #1 to #3*** *and finish it no later than* ***deliverable and milestone #4****.]*

* *Team members are expected to have a working knowledge of Java, MySQL, and other relevant technologies*
  + *Solutions: The team will aim to improve their knowledge of the Java and JavaFX languages to make the application better. In case of problems, the team will try to work together to solve the issue and help each other.*
* The project schedule and the resources assigned may be impacted by unanticipated changes in the project requirements or scope
* When the user base expands, the program may have performance bottlenecks or have trouble scaling
  + Solutions: If such scenario occurs, the team will significantly improve the DB
* Team member leaves the project
  + Solutions: Either the rest of the team will have to take up the work of the team member, or give up on certain implementations and functionalities due to time constraints.
* Uncooperative/slacking team member
  + Solution: The rest of the team will take that team member’s responsibilities for the sake of finishing the project and delivering it on time
* Bugs and errors
  + Solutions: Through the code the team will try to anticipate potential errors and bugs that may arise from the application’s features. Validation and checking for values will be implemented where there is a risk of an error or exception from occurring. However, as it is the case with bugs and errors, one cannot anticipate all of them without rigorous testing. Therefore, the team will dedicate time to each feature to test whether they work properly and if they are err free.

# Requirements

*[This chapter should be started for the* ***deliverable and milestone #0****, and some parts (context and functional requirements) should be finished for the* ***deliverable and milestone #1****. However, some parts of it will be produced and/or changed later in* ***deliverables and milestones #2, #3, and #4****.]*

## Context

*[Provide a description of the application in the broader context, how it will work within the environment of other systems (e.g., payment systems if there is some kind of purchasing involved), with explanations as applicable. The context of a system refers to the connections and relationships between the application and its environment. At least several sentences are expected and it relates to* ***deliverables and milestones from #1 to #5****.]*

The application will be desktop run. If monetization will be necessary, it could be through advertisements. The goal of the application is to attract as many users as possible by being widely accessible for all; with that in mind, we would prefer for the application to be free to use and download.

Users will be able to utilize the FilmFinder without having to go through the hassle of logging in, as some users may not want to make an account in the application. For users that only wish to view the movies, actors and directors of movies, they may do so without logging in. Users who log in will be granted special privileges to use the application’s additional features that are only available to logged and registered users.

## Functional Requirements

*[****IMPORTANT:*** *List, name and explain all key functionalities – there should be approx. 15 functional requirements listed. You should provide a table where each requirement is named (short code or name, could combine letters with numbers) and a detailed description, as well as who is responsible or performing the action associated with this functional requirement. You could also use use-cases (diagrams or descriptions), including use-case names, actors, events flow, exceptions, and special requirements. Include user requirements if necessary (users, roles, privileges) and associate with specific functional requirements. This should be started for the* ***deliverable and milestone #0****, and the proposal must be agreed upon and finished with* ***deliverable and milestone #1****. Later, with the approval of the client, it could be revised in* ***deliverables and milestones #2 - #4****).]*

The final product should have the following requirements:

1. **User Authentication**: The application must have a user authentication system to ensure that only registered users can access the features of the application.
2. **Movie and TV Series Database**: The application should have a comprehensive database of movies and TV series, including details such as title, genre, year of release, cast, synopsis, images and user ratings.
3. **Search and Filter Functionality**: The application should provide users with the ability to search and filter movies and TV series based on various criteria such as title, genre, year, rating, and other relevant factors.
4. **User Interaction**: The application should provide an interactive platform for users to contribute to the database by adding, editing, and deleting movies and TV series.
5. **User Profile**: The application should allow users to create and manage their profiles, which should include details such as username, password, email address, and, potentially, preferences.
6. **Admin Panel**: The application should have an admin panel that allows the administrator to manage user accounts, movie and TV series database, and other application settings.
7. **Security**: The application should implement robust security measures to ensure that user data is protected from unauthorized access. Preventive measures against various security issues such as SQL injections should be implemented.
8. **Compatibility**: The application should run on any PC platform which is able to run and server Java-based applications.
9. **Performance**: The application should be optimized for performance, with fast search and loading times and minimal downtime.

| ***NUMBER*** | ***REQUIREMENT CODE*** | ***DESCRIPTION*** | ***ACTOR*** |
| --- | --- | --- | --- |
| *1* | *Functional requirement 1* | *Search for movies/series by title, director, actor, genre* | *User* |
| *2* | *Funct. req 2* | *View detailed information about a movie/series, including synopsis and cast (directors, actors etc.)* | *User* |
| *3* | *Funct. req 3* | *Add, edit, and delete movies/series from the DB* | *Admin* |
| *4* | *Funct. req 4* | *Add, edit, and delete directors* | *Admin* |
| *5* | *Funct. req 5* | *Register a new user account* | *User/Admin* |
| *6* | *Funct. req 6* | *Login with an existing account* | *User/Admin* |
| *7* | *Funct. req 7* | *View user account information* | *User/Admin* |
| *8* | *Funct. req 8* | *Submit reviews and rating for movies/series* | *System* |
| *9* | *Funct. req 9* | *View user reviews and ratings for movies/series* | *User* |
| *10* | *Funct. req 10* | *Update user account information* | *Admin/User* |
| *11* | *Funct. req 11* | *Add informations, edit and delete for Actors table* | *Admin* |
| *12* | *Funct. req 12* | *Add informations, edit and delete for Genres table* | *Admin* |
|  |  |  |  |

## Other (Non-Functional) Requirements

*[Describe the non-behavioral and non-functional requirements, including hardware and software requirements (e.g., platforms needed to support this application), programming interfaces, and any operational requirements (how the system will run and communicate with the environment). You could also provide information about application availability (time of day or week), general performance (how fast it should be in client responses), capacity (how many concurrent users or connections it will support), error handling (how is it handled), conventions used, security and similar if necessary. This should be started for the* ***deliverable and milestone #1****, and it could be revised in* ***deliverables and milestones #2 - #5****).]*

* The application should utilize a RESTful API for client-server communication
* The application should be designed to run on standard personal computers and mobile devices with an internet connection
* The application should aim for a 99.5% uptime, allowing for occasional maintenance or unforeseen issues
* The application should gracefully handle errors, providing meaningful error messages to the user when necessary
* The system should log errors for developers to analyze and resolve issues
* The application should implement proper authentication and authorization mechanisms to protect user data
* The application should be available 24/7, barring any scheduled maintenance periods

# User Documentation

## Graphical User Interface Design

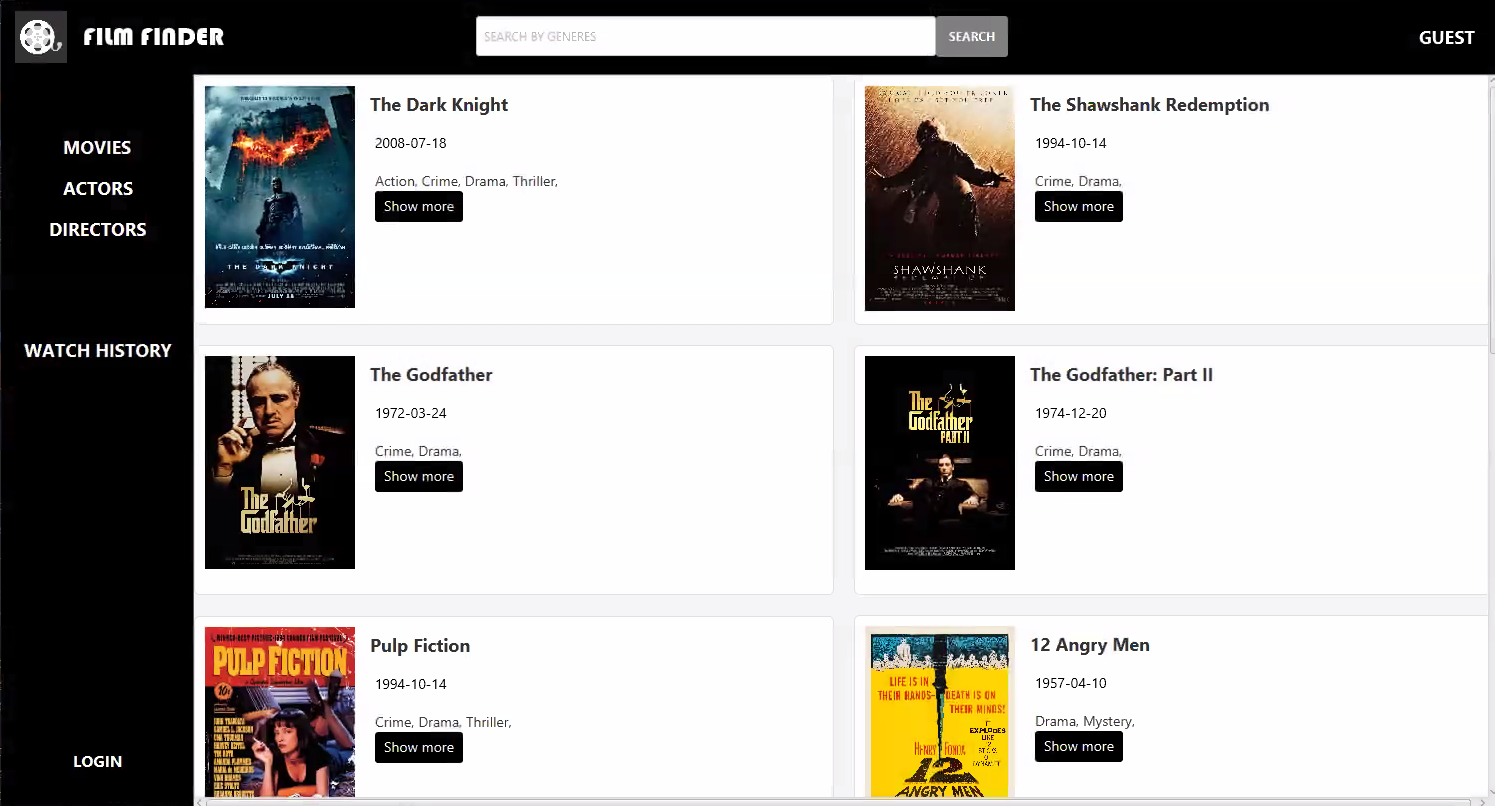
*[You should provide user design and user experience description, as well as a description of used technologies. This should be started for the* ***deliverable and milestone #3****, and should be finished with* ***deliverable and milestone #4****.]*

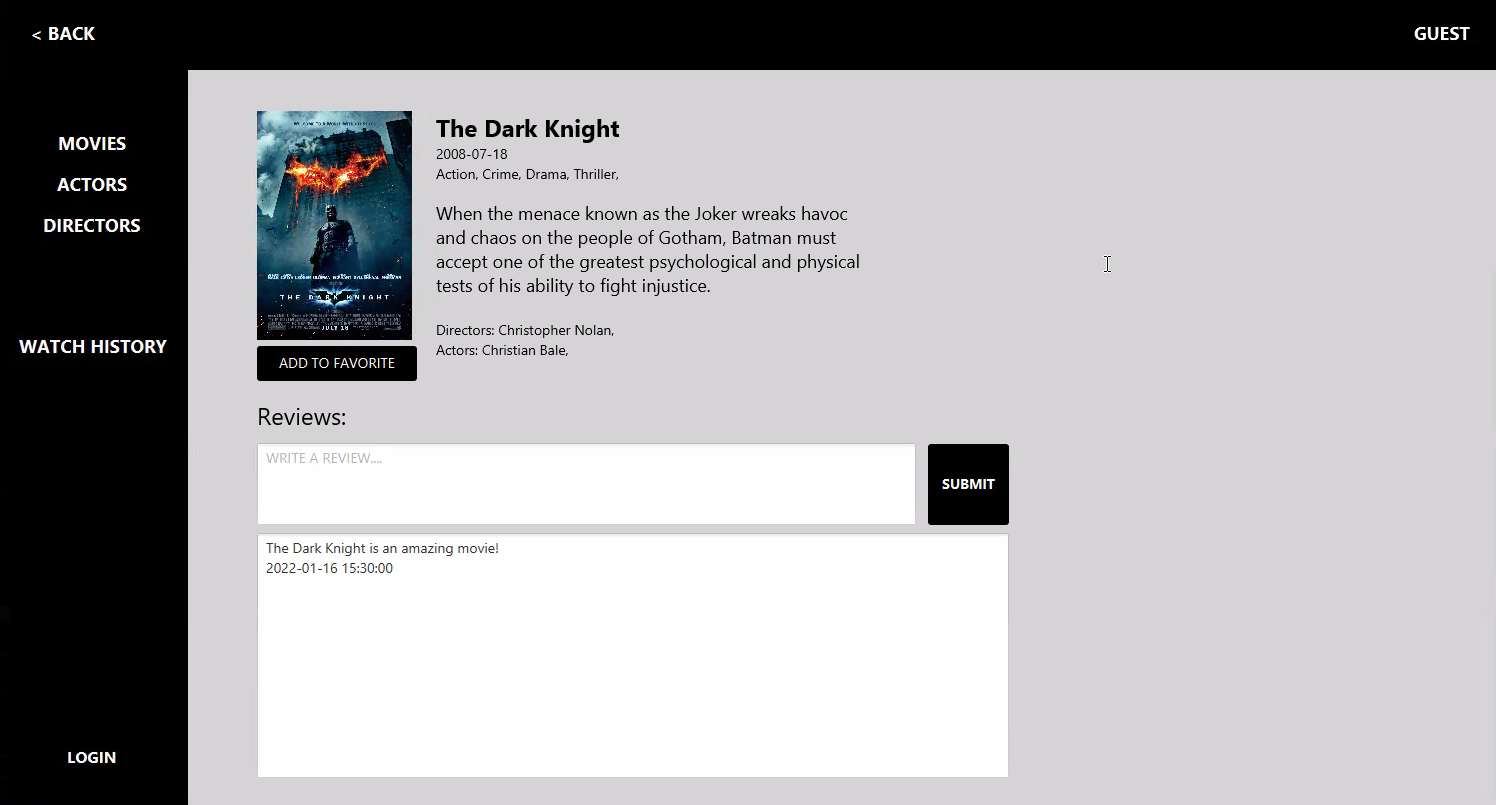
*The GUI of the FilmFinder application features a clean and user-friendly design. The dark background color provides a sleek and modern look, while the search bar and grid or list view for movies and series make it easy for users to navigate and find what they're looking for.*

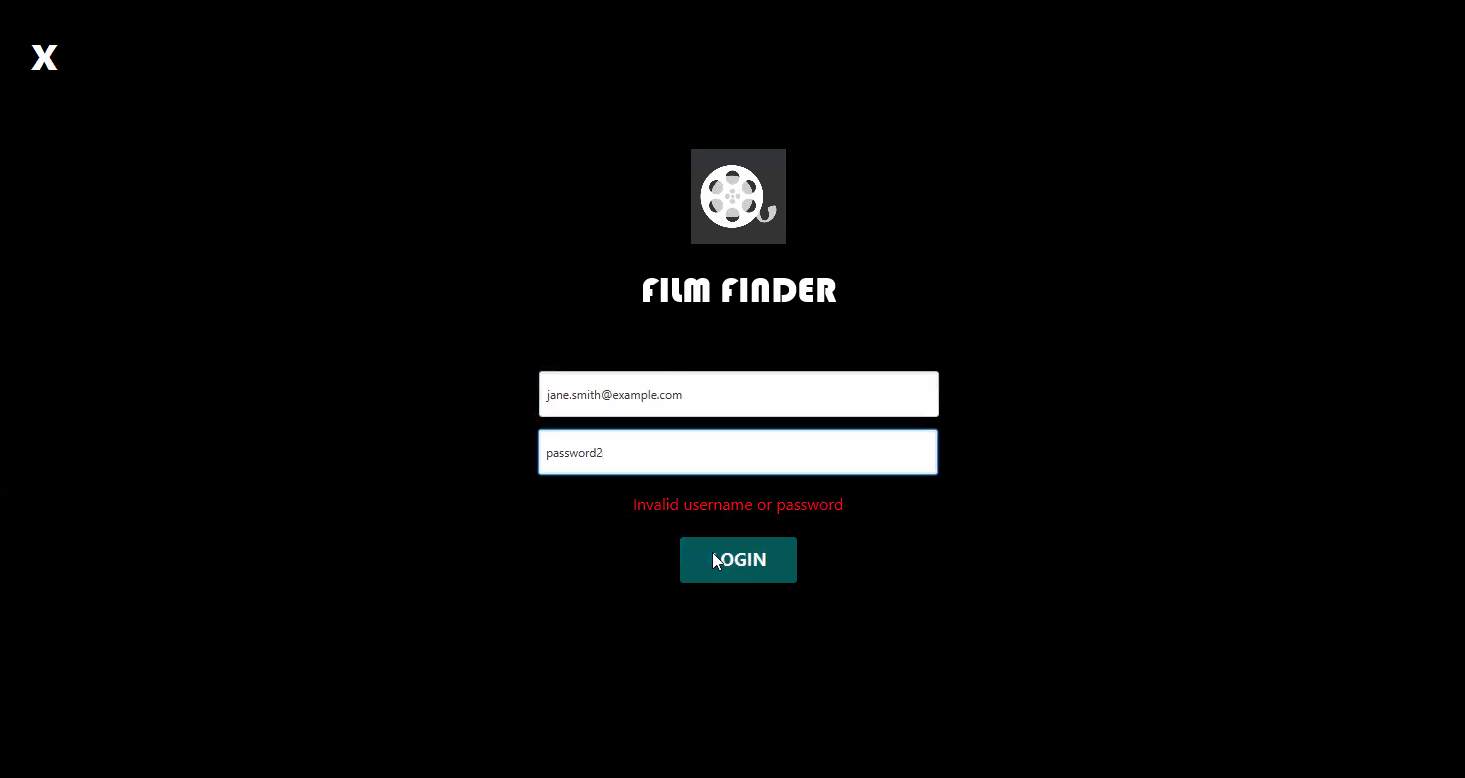
*Upon launching the application, the user will be directed to the movie search page, where they can start exploring the database. It is recommended for the user to log in, and clicking on the login button will open the login page. The login page includes two input fields for email/username and password, with a Login button below them.*

*In addition to the movie search page, the application also includes sections for Movies, Actors, and Directors. Each section displays images and other relevant information below them, providing an intuitive and visual way for users to browse and discover content. The layout and structure of these sections are designed to optimize user experience and facilitate seamless navigation between different parts of the application.*

*Technologies used in the development of the application include JavaFX for the GUI, as well as back-end technologies for database management and communication with the business layer.*







## User Manual

*[This should provide expected usage of the available functionalities, could be divided per user roles, and should include screenshots with detailed descriptions. This should be started for the* ***deliverable and milestone #3****, and should be finished with* ***deliverable and milestone #4****.]*

**Admin Role:**  
As an admin, you have access to all the functionalities of the platform. You can manipulate data by adding, editing, or deleting movies or shows. To add a new movie or show, click on the "Add new" button on the dashboard.

You can also review movies. To review a movie, click on the “Show more” button for a movie entry, and you will be greeted with a input area for your review. After finishing writing, you can submit it. To search for a movie, click on the "Search" button on the dashboard.

**User Role**:  
You can comment on movies. A comment box will appear, and you can type in your comment. Once you're done, click on the "Submit". You will also see the previous comments. To search for a movie, click on the "Search" button on the dashboard. You can use a watchlist. The user is similar to the Guest, but unlike the guest, the user can leave reviews and have access to a watch history.

**Guest Role:**As a guest, you can only search for movies or shows. Click on the "Search" button on the dashboard, type in the title, genre, and the platform will display a list of movies or shows that match your search criteria. A guest is a user who has not logged into the application.

# Installation, Configuration and Acceptance Testing

*[Usually, this chapter should be started later, and at least partially filled with* ***deliverable and milestone #3 or #4****, and should be finished with* ***deliverable and milestone #5****.]*

## Installation

*[Provide a technical manual – prerequisites and installation process description details. Should be finished with* ***deliverable and milestone #5****.]*

**Prerequisites:**

Java SE Development Kit (JDK) 11 or higher

MySQL Server 8.0 or higher

**Installation Steps:**

* Download the FilmFinder installation package from the "provided link".
* Extract the package to a desired directory on your local machine.
* Check if Java SE Development Kit 11 or higher is installed on your machine by running the following command in the terminal: java -version. If Java is not installed or the version is below 11, please download and install the latest version of Java SE Development Kit.
* Check if MySQL Server 8.0 or higher is installed on your machine and running by logging into the MySQL Server: mysql -u root -p. If MySQL Server is not installed or not running, please download and install the latest version of MySQL Server.
* If the application fails to connect to the database, ensure that the database is set up correctly and the connection details in the "application.properties" file are correct.
* Open a terminal or command prompt and navigate to the directory where the FilmFinder application was extracted.
* Run the following command to launch the application: "java -jar FilmFinder.jar".
* The login screen should appear. Enter your login credentials or create a new account to access the application.

## Configuration

*[Technical manual should also hold configuration detail and default values for this project to work. Should be finished with* ***deliverable and milestone #5****.]*

To ensure the successful installation and configuration of this project, the following technical details and configuration information are provided:

Java version:  
This project was developed using Java version 17, but it should also work with Java version 11.

MySQL Workbench version:  
This project was developed using MySQL Workbench version 8.0. Please ensure that you have MySQL Workbench 8.0 installed on your system to run this project.

JavaFX version:  
This project uses JavaFX version 17, which must be installed on your system before running the project. Please ensure that you have the correct version of JavaFX installed.

Configuration of the connection:

After MySQL has been installed. To ensure that the MySQL connection works correctly, within the “lib/dbConfg.txt” file, change the parameters for the three lines present in the text file. Depending on what the user has set as their MySQL username and password, change the values, after the “=” sign to the password and username set up for the MySQL.

By default this is what is present in the dbConfig.txt file:

“jdbc.url=jdbc:mysql://localhost:3306/filmfinder

jdbc.username=root

jdbc.password=password

“

## Acceptance Testing

**Data Access Layer Testing**

Tested out the CRUD for all models. All are working correctly.

Genres.java model - Added a checking method to see if user tries to insert or update an already existing Genre name into the

Review & Watch\_History.java - Made sure that the Timestamp is properly formatted when fetching, updating and inserting

Deletion and foreign keys - If a table’s id is a foreign key in another table, then the id is beforehand deleted from those tables, before the main table.

**Database**

Tested out the inserting.

The entire .sql file can be run without any issues.

**Business Layer**

All methods are functional.

Logging in works, and logging out as well.

# Final Remarks and Conclusion

**Positive experiences:**Working on a group programming project can be a great opportunity to meet new people who share your passion for programming. Collaborating with new team members can help you learn how to work effectively with others and contribute to the project's success. Learning to collaborate on a project is an important skill that can help you communicate more effectively, share ideas, and solve problems together. Seeing the project grow week by week and the result of your hard work can give you a great sense of pride and accomplishment.

**Negative experiences:**  
Working on a group programming project can also have some negative experiences. An unexpected withdrawal of a team member can be a setback and may require you to reorganize the team and adapt to new circumstances and we adapt well to this issue. Spending long nights in front of a computer instead of in bed can lead to burnout and negatively impact your overall well-being. Technical challenges and tight deadlines can be frustrating and stressful, requiring additional research, testing, and debugging.

**Concerns**:  
While working on a group programming project, several concerns may arise. One of the main concerns is whether the project team will receive enough points to pass. It is also essential to ensure that the team members are satisfied with each other's work and communicate effectively to achieve project success. Moreover, the professor's satisfaction with the team's work is crucial for academic success. Finally, the product of the team's work must be functional and of high quality to ensure its usefulness in real-world scenarios.

**Missing functionality:**The missing movie collection functionality can be a significant drawback for users who want to keep track of the movies they've watched or want to watch. Unfortunately, it seems that the team realized too late in the project that the database would need to be rewritten to accommodate this feature.

Although it's disappointing that this functionality could not be implemented, the team can take this as a learning opportunity to identify the issues that caused this setback and brainstorm ways to address them in future projects. For example, during the planning phase of the project, the team can conduct more in-depth research on the technical requirements for each functionality to avoid discovering issues too late in the process.

In terms of possibilities for improvement and extensions, there are a few potential ideas to consider. One option could be to build a separate movie collection app that could integrate with the streaming platform. This would allow users to keep track of their movies and shows separately and provide an added convenience. Another possibility would be to implement a watch history feature that tracks what movies and shows users have already watched. This would be a simpler feature to implement and would still provide value to users.

Ultimately, the missing movie collection functionality can serve as a valuable lesson for future projects, highlighting the importance of thorough planning and research to avoid setbacks later in the process. While it's disappointing not to have this feature, there are still other possibilities for improvement and extension that the team can consider.

**Possibility for improvement**:  
While the inability to add the watchlist feature may be disappointing, it's important to explore possibilities for improvement and extensions to the project. There are several potential ideas that can be implemented to take the project to the next level and offer a more comprehensive and user-friendly streaming platform.

The most obvious possibility is to add the watchlist feature to the project. This would give users the ability to save movies or TV shows they're interested in and watch them later. By implementing a recommendation engine, users can receive personalized recommendations based on their viewing history and saved movies or shows on their watchlist.

Integrating social media features can also enhance the user experience. Users can share their viewing habits and watchlists with friends and receive recommendations from their social circle. Additionally, by allowing users to leave reviews of the movies or shows they've watched, other users can read reviews before watching something new, helping them make an informed decision about what to watch.

Improving the search function can also make a significant difference in the user experience. By enhancing the search function, users can easily find new movies and shows based on criteria such as genre, year of release, and actor or director. These ideas for improvement and extensions can help the team take the project to the next level and offer a more comprehensive and user-friendly streaming platform.

**Estimate effort:**  
Karlo spent 50 person-hours working on the GUI and documentation, while Filipa spent 70 person-hours working on the data access layer, business layer, database and database layer, and documentation. All team members contributed to documentation, recognizing its importance for the project's success.  
  
At the beginning of the project, the team recognized that the business layer could be the biggest issue. To address this challenge, Fran and Vid decided to work on it together, dedicating a total of 90 person-hours to this aspect of the project.

Although the total project effort is estimated at 210 person-hours, the time distribution for each team member's work is not specified, making it difficult to determine how the effort was distributed over the course of the project. However, more detailed information on the distribution of work will be provided in the self-evaluation and peer review process, where each team member will have the opportunity to reflect on their contributions to the project and provide feedback on their peers' contributions.

By working collaboratively and recognizing each other's strengths and weaknesses, the team was able to overcome challenges and complete the project successfully. The self-evaluation and peer review process will allow the team to assess the project effort and distribution of work accurately, providing valuable insights for future projects.