

# Projetct ISPW

## INDEX

1. SOFTWARE REQUIREMENT SPECIFICATION
  - 1.1. INTRODUCTION
    - 1.1.1. AIM OF THE DOCUMENT
    - 1.1.2. OVERVIEW OF THE DEFINED SYSTEM
    - 1.1.3. HARDWARE AND SOFTWARE REQUIREMENTS
    - 1.1.4. RELATED SYSTEMS, PROS AND CONS
  - 1.2. USER STORIES
    - 1.2.1. US-1
    - 1.2.2. US-2
    - 1.2.3. US-3
  - 1.3. FUNCTIONAL REQUIREMENTS
    - 1.3.1. FR-1
    - 1.3.2. FR-2
    - 1.3.3. FR-3
  - 1.4. USE CASES
    - 1.4.1. OVERVIEW DIAGRAM
    - 1.4.2. INTERNAL STEPS
2. STORYBOARDS
3. DESIGN
  - 3.1. CLASS DIAGRAM
    - 3.1.1. VOPC (ANALYSIS)
    - 3.1.2. DESIGN-LEVEL DIAGRAM
  - 3.2. DESIGN PATTERNS
  - 3.3. ACTIVITY DIAGRAM
  - 3.4. SEQUENCE DIAGRAM
  - 3.5. STATE DIAGRAM
4. TESTING
5. EXCEPTIONS
6. DATA BASES
  - 6.1. FILE SYSTEM
  - 6.2. MYSQL
7. SONAR CLOUD

# 1. SOFTWARE REQUIREMENT SPECIFICATION

## 1.1 INTRODUCTION

### 1.1.1. AIM OF THE DOCUMENT

This document objectively presents the fundamental and specific requirements of the system. It reviews key internal and external aspects, supported by relevant diagrams, to provide a comprehensive understanding.

### 1.1.2. OVERVIEW OF THE DEFINED SYSTEM

The system is an application designed to help users organize their shows into personal lists and view meaningful statistics about these lists. It retrieves show data from an external API, enabling users to search for almost any show and access its information to generate their personalized lists.

### 1.1.3. HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

- A computer capable of running JAR files.
- Sufficient disk space to store user data, particularly if the offline version is utilized.

Software Requirements:

- Oracle JDK SE 24 (or compatible JRE) for proper execution of the JAR file.
- A DBMS server (either remote or local) to host the database, allowing the online version of the system to retrieve information via SQL queries.

### 1.1.4. RELATED SYSTEMS, PROS AND CONS

To contextualize the proposed system, it's beneficial to examine existing platforms that offer similar functionalities. Notable examples include:

#### **Trakt.tv**

Trakt.tv is a popular community-driven platform for tracking movies and TV shows, offering features like syncing with various streaming services and engaging with a community of media enthusiasts.

Pros:

- Custom List Creation: Enables users to create and share personalized lists of favorite shows, watchlists, or themed collections.
- Calendar & Notifications: Tracks air dates for upcoming episodes and movie releases.
- Stats & History: Provides detailed insights into viewing habits, including total watch time and genre breakdowns.
- Personalized Recommendations: Suggests shows and movies based on watch history and user ratings.

Cons:

- Inconsistent Syncing: Some integrations (e.g., Netflix) rely on third-party applications that may not always function perfectly.
- Learning Curve: New users may initially find the interface and extensive integrations somewhat complex.

## **Simkl.com**

Simkl is a platform that helps users track movies, TV shows, and anime across various platforms.

Pros:

- **Comprehensive Tracking:** Keeps track of TV shows, anime, and movies across multiple streaming services.
- **Custom Lists & AI-powered Recommendations:** Allows users to create custom watchlists and receive personalized recommendations.
- **Cloud-Based History:** Securely stores watch history, ensuring users don't lose track of their viewing progress.

Cons:

- **Occasional Sync Issues:** Some users report problems with syncing data, particularly with services like Netflix.
- **Potentially Cluttered UI:** The interface may be perceived as overwhelming or outdated by some users.

## **1.2. USER STORIES**

### **1.2.1. US-1**

**As a User, I want** to search and view detailed movie information **so that** I can learn more about them before watching.

### **1.2.2. US-2**

**As a User, I want** to create and manage personal lists of movies **so that** I can keep track of the ones I have watched and want to watch.

### **1.2.3. US-3**

**As a User, I want** to track the total time I've spent watching movies **so that** I can monitor my viewing habits.

## **1.3. FUNCTIONAL REQUIREMENTS**

### **1.3.1. FR-1**

The system shall display movie details, including title, release year, genre, and overview.

### **1.3.2. FR-2**

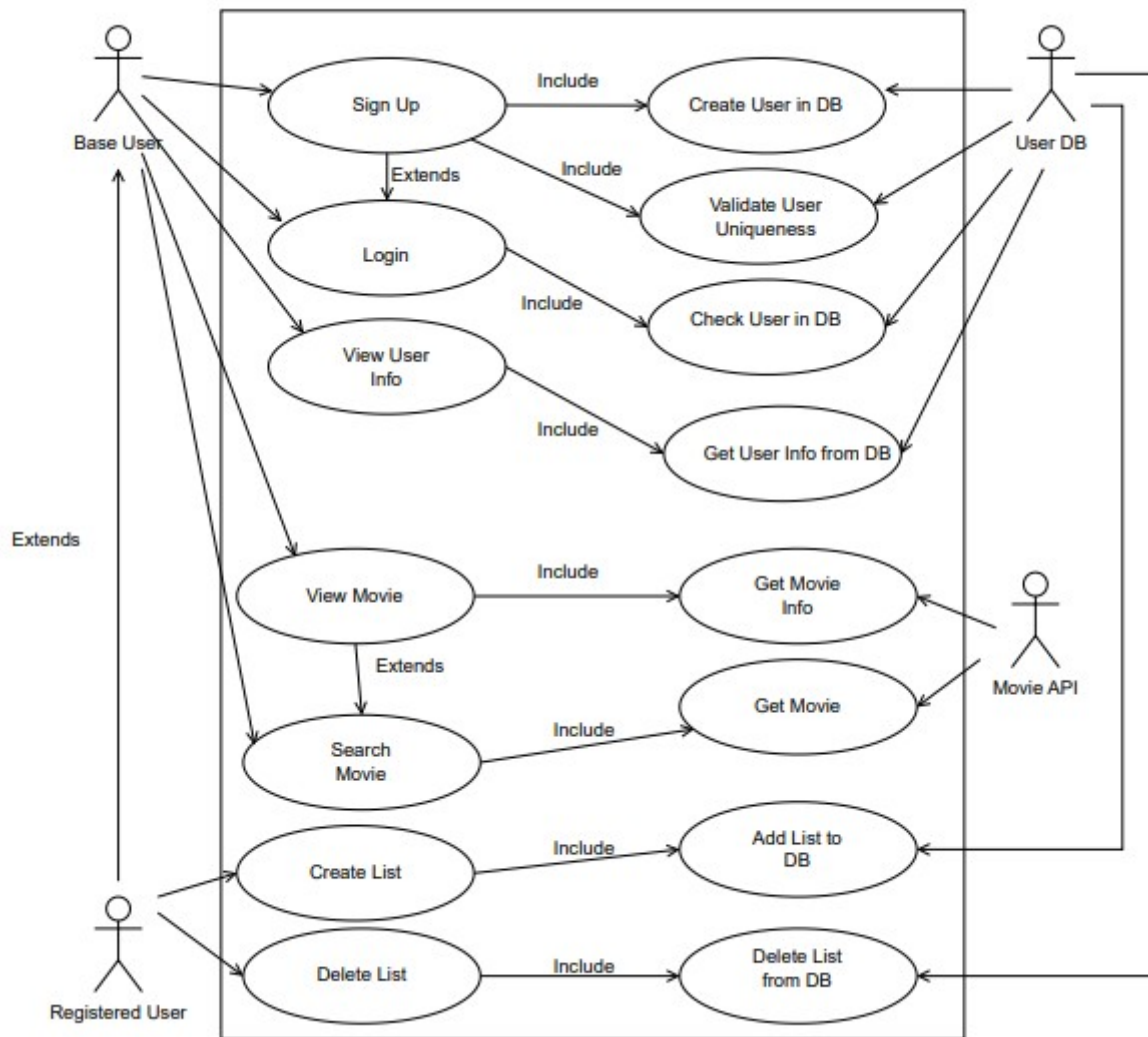
The system shall allow users to add and remove movies from their lists.

### **1.3.3. FR-3**

The system shall calculate and display the total watch time for a user-created list of movies.

## 1.4. USE CASES

### 1.4.1. OVERVIEW DIAGRAM



### 1.4.2. INTERNAL STEPS

#### Use Case: Add Movie to personal list

1. User Request: The user initiates the request to add a movie to a personal list.
2. System Displays Lists: The system presents the user with a list of their existing personal lists.
3. User Selects List: The user selects an existing personal list or chooses an option to create a new one.
4. System Confirms Selection: The system acknowledges the selected list.
5. User Confirms Addition: The user confirms that the selected movie should be added to the chosen personal list.
6. System Validates Uniqueness: The system verifies that the movie is not already present in the selected personal list.

7. System Saves Movie: The system saves the movie to the selected personal list in the database.
8. System Notifies Success: The system displays a confirmation message indicating that the movie has been successfully added to the list.

### Extension

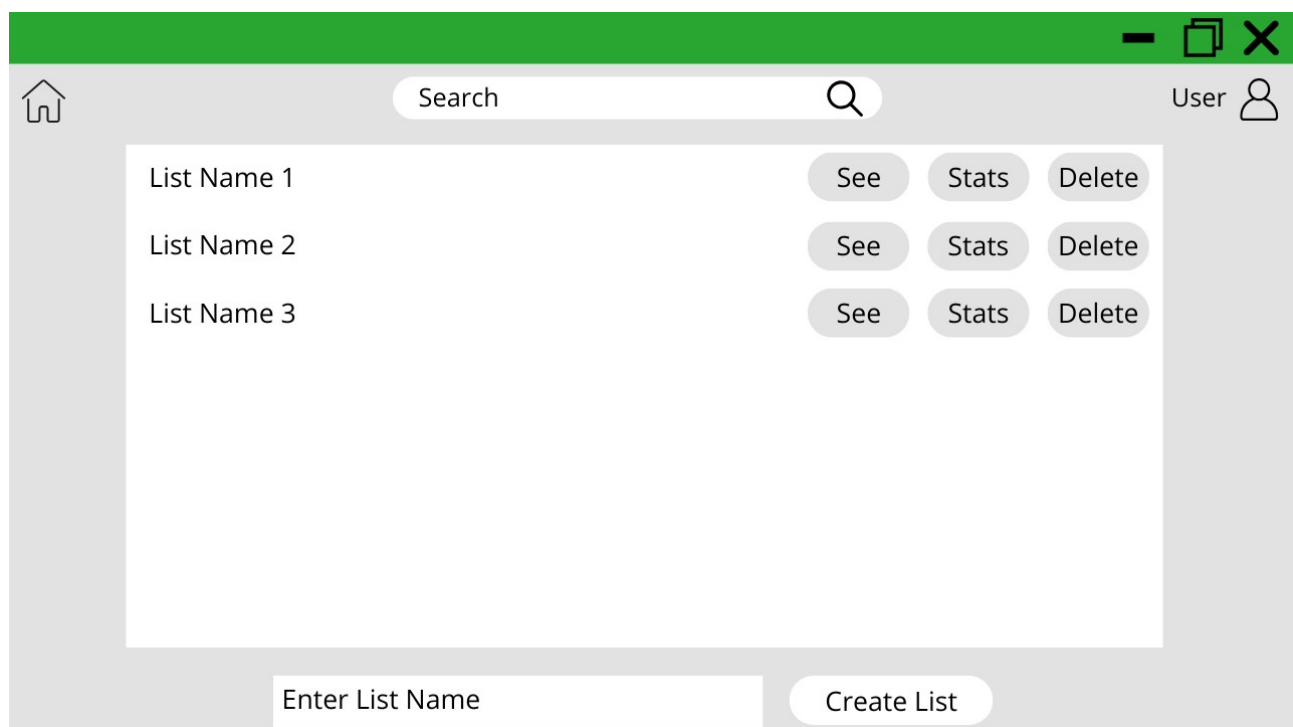
1a. User chooses to create a new list: The system prompts the user to name the new list and after the user enters a name for the new list the system creates the new list and proceeds to step 4 (System Confirms Selection) with the newly created list.

6a. Movie already in the selected list: The system notifies the user that the movie is already in the selected list.

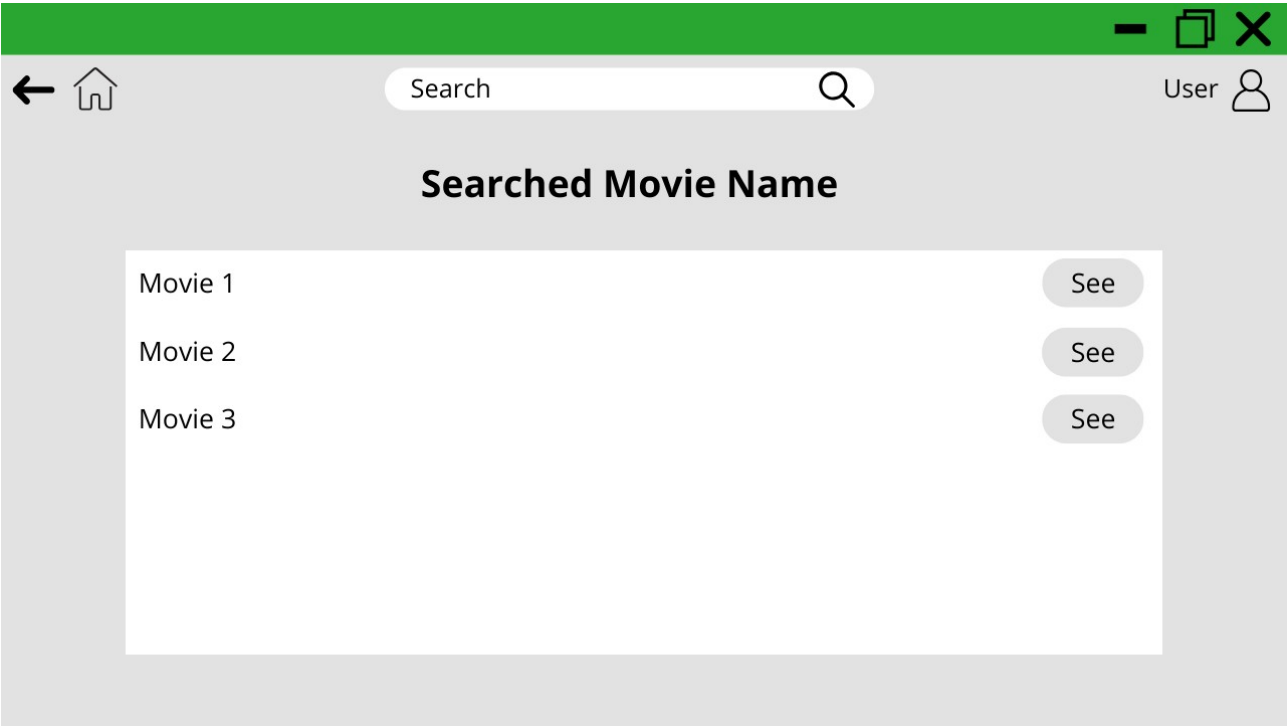
7a. Database Save Error: The system encounters an error while saving the movie to the database and the system notifies the user of the error.

## 2. STORYBOARDS

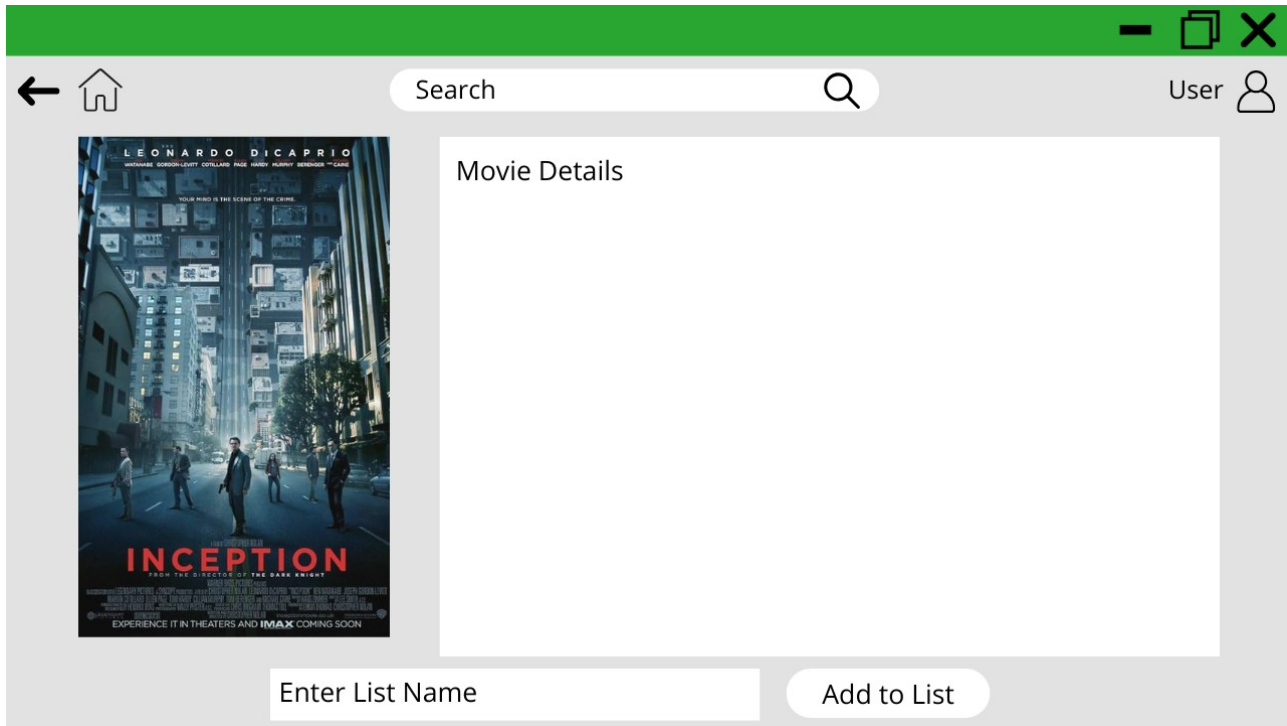
This storyboard is the home page where the user can see his personal lists if it is logged and manages the lists. The user can search for movies.



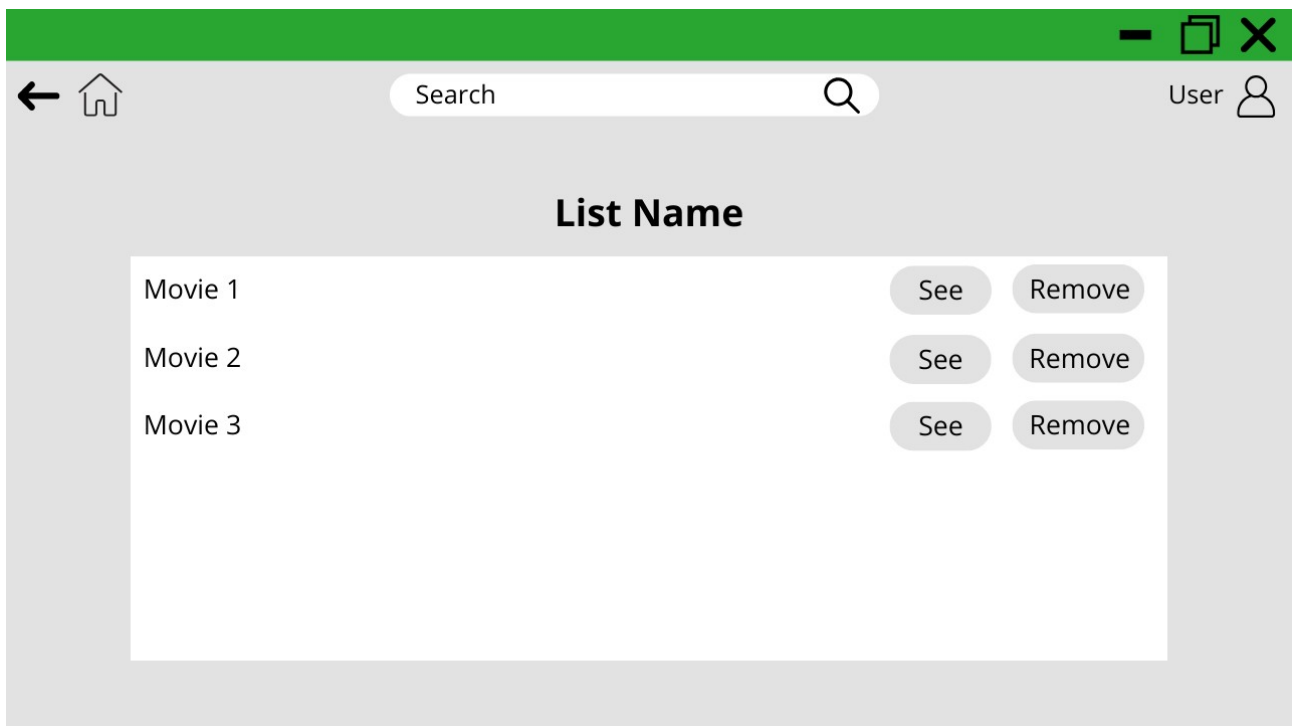
This storyboard is the search page. After the user searched for a movie the system display a list of movies.



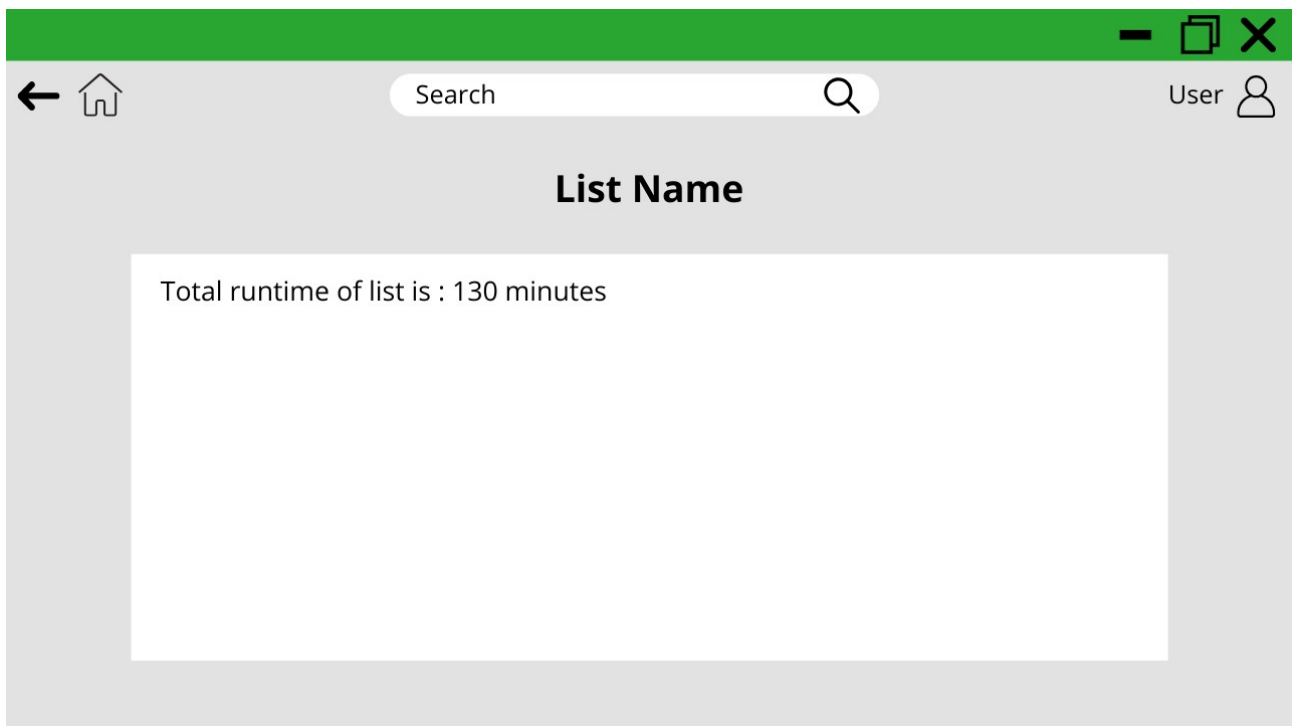
This storyboard is the show page, where movie informations are displayed and the user can add the movie to a list.



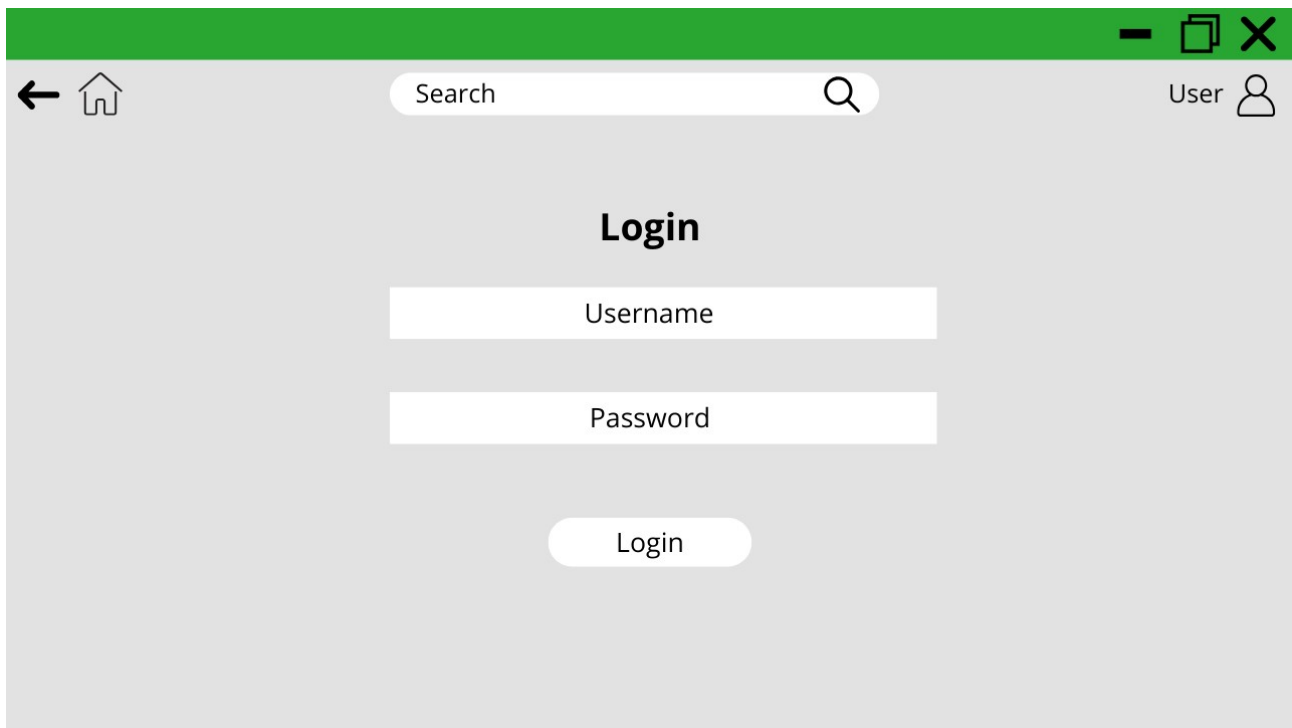
This storyboard is the list page where are displayed the movies of a list.



This storyboard shows the stats of a list.

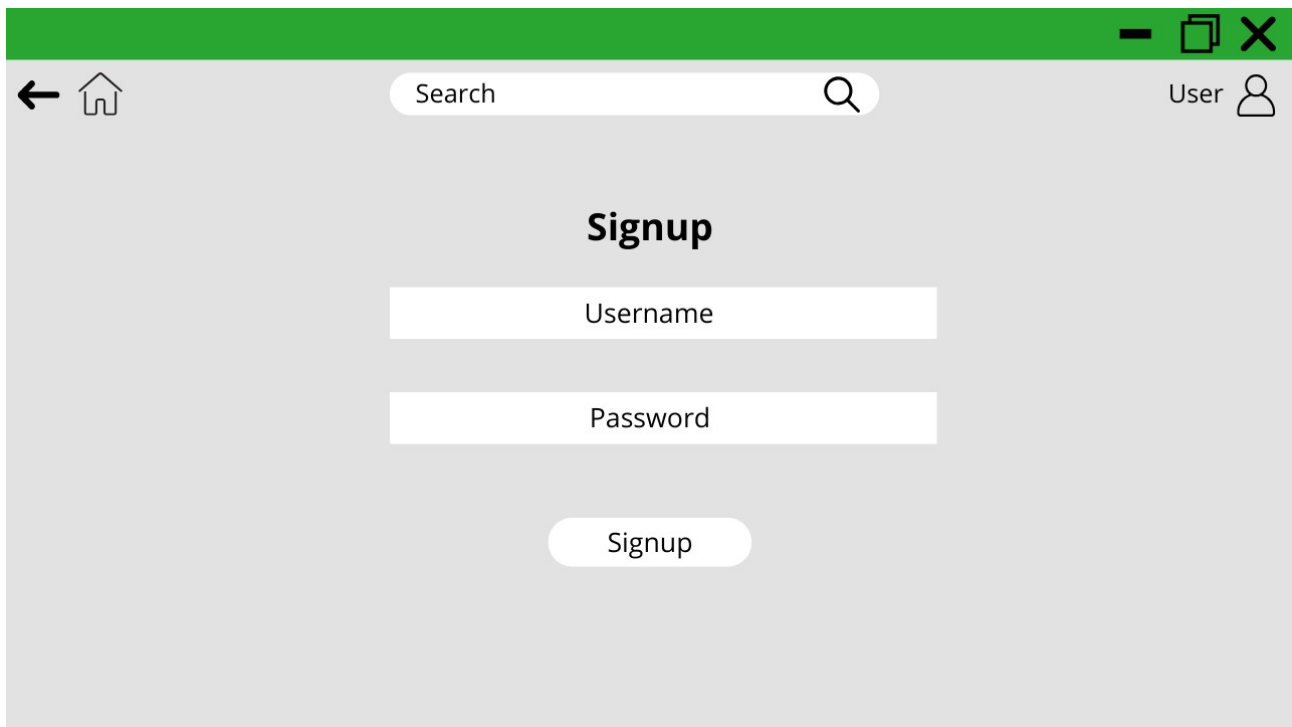


This storyboard is the login page.



A storyboard for a login page. The top bar is green and contains a minus sign, a square icon, and a close 'X' icon. Below the top bar is a navigation bar with a back arrow, a home icon, a search bar with the text 'Search' and a magnifying glass icon, and a user profile icon with the text 'User'. The main content area is light gray and contains the title 'Login' in bold. Below the title are two white input fields: 'Username' and 'Password'. At the bottom is a white rounded button labeled 'Login'.

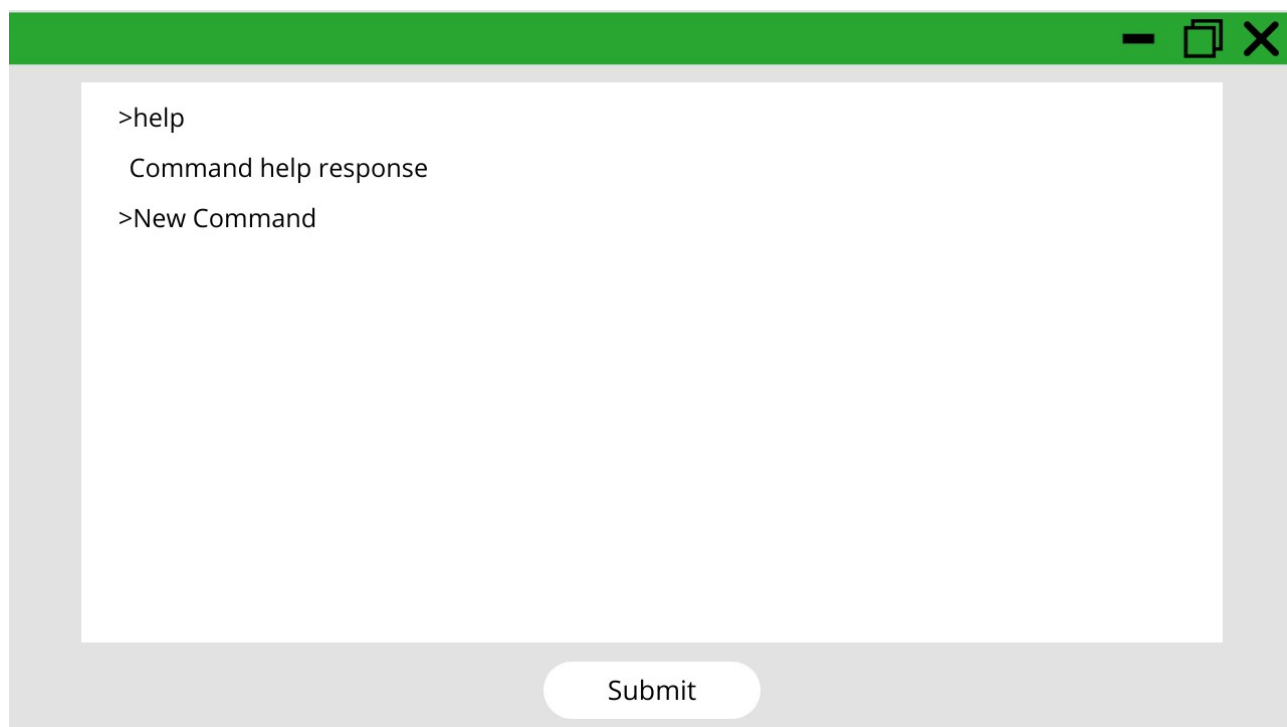
This storyboard is the signup page.



A storyboard for a signup page. The top bar is green and contains a minus sign, a square icon, and a close 'X' icon. Below the top bar is a navigation bar with a back arrow, a home icon, a search bar with the text 'Search' and a magnifying glass icon, and a user profile icon with the text 'User'. The main content area is light gray and contains the title 'Signup' in bold. Below the title are two white input fields: 'Username' and 'Password'. At the bottom is a white rounded button labeled 'Signup'.



This storyboard is a cli where the user can writes commands to search movies and manages personal lists.



The storyboard depicts a CLI application window with a green title bar and standard window controls. The main content area is white and contains the following text:

```
>help  
  Command help response  
>New Command
```

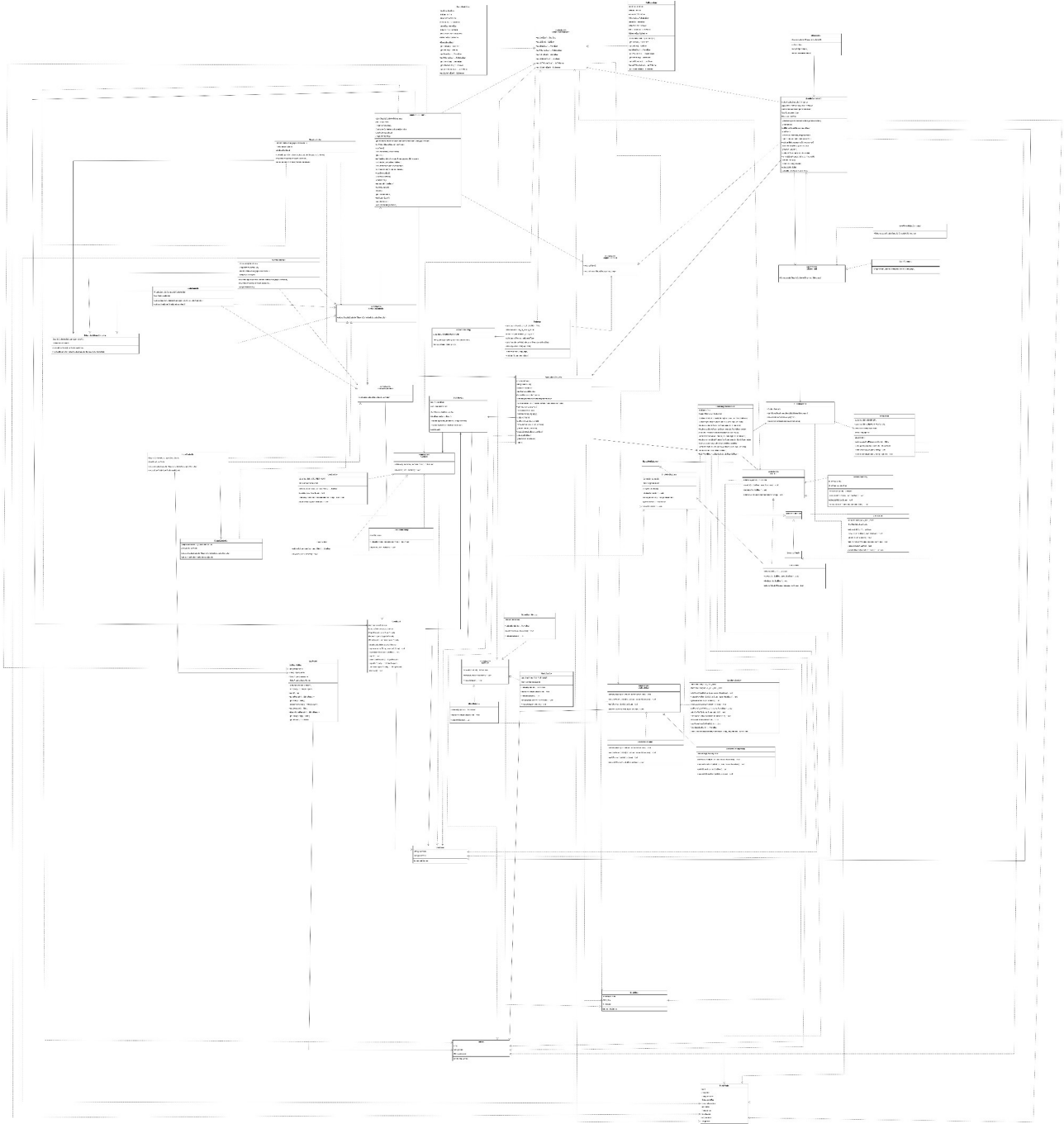
At the bottom center of the window is a rounded rectangular button labeled "Submit".

### 3. DESIGN

#### 3.1. CLASS DIAGRAM

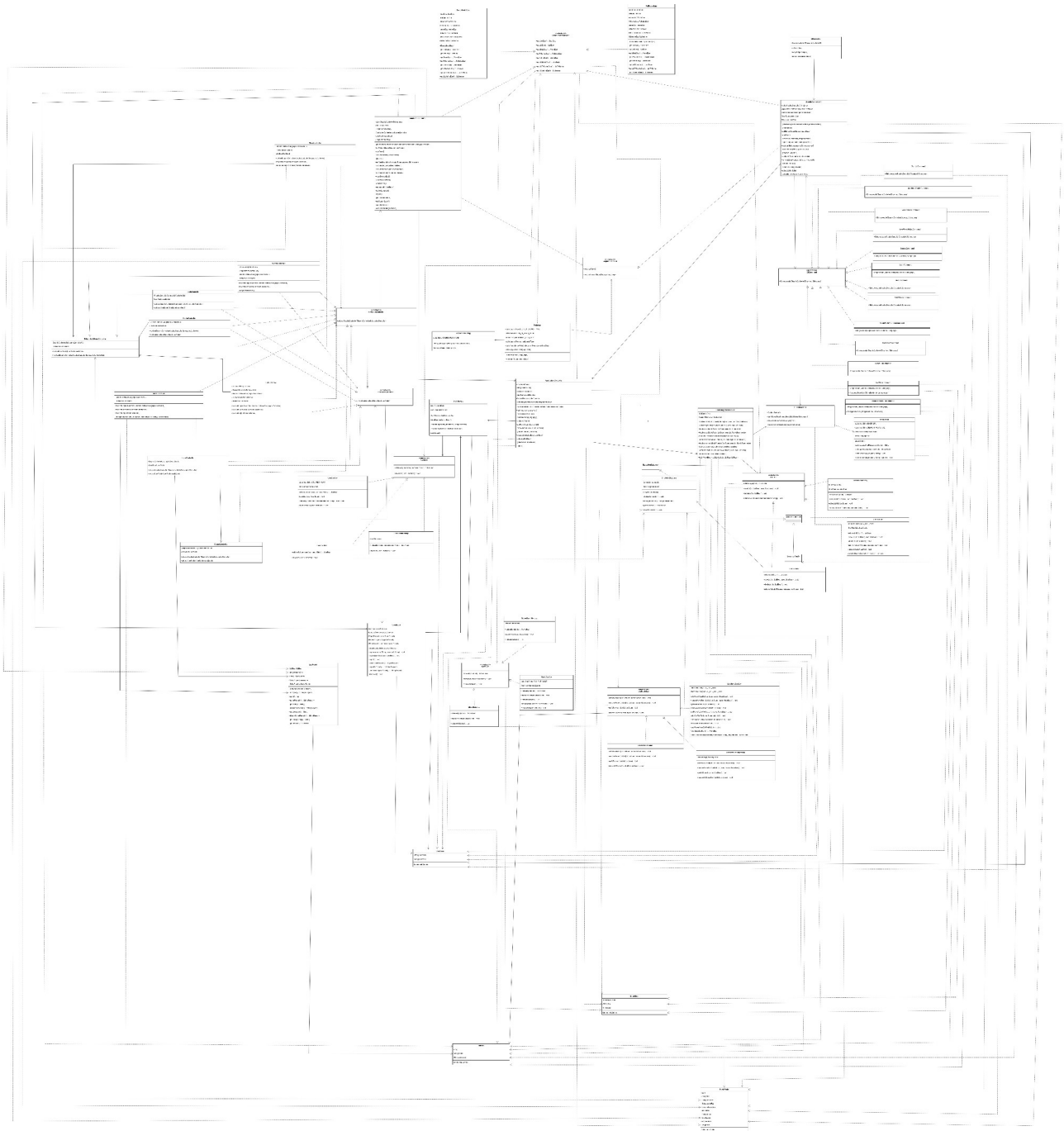
##### 3.1.1. VOPC (ANALYSIS)

This diagram is attached near to this document to be able to have a closer look to it.



3.1.2. DESIGN-LEVEL DIAGRAM

This diagram is attached near to this document to be able to have a closer look to it.



### 3.2. DESIGN PATTERNS

In the context of the system, the **Singleton** pattern has been implemented, primarily through the **SingletonDatabase** class. The main point of this pattern is to ensure that there is one and only one instance of a particular type of object, so all actions related to that object are correctly executed without incoherence.

The SingletonDatabase class serves as a singleton for the database connection. This design was a significant help because it provides a single, consistent point of access to the database.

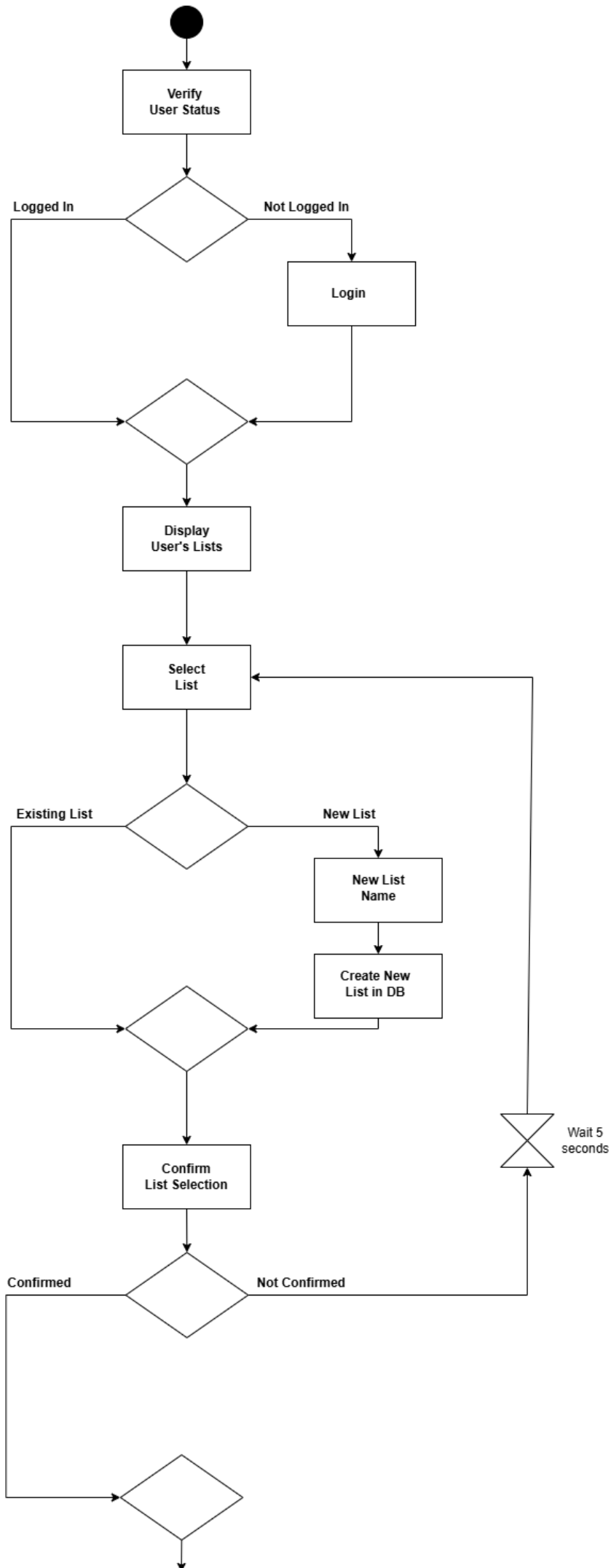
Consequently, the DAO (Data Access Object) classes (JDBC ones) that need to access information from the database do not need to create their own instances of the database connection or have it passed as a parameter in every constructor. Instead, they can rely on the single instance provided by SingletonDatabase, simplifying their design and ensuring consistent database interaction.

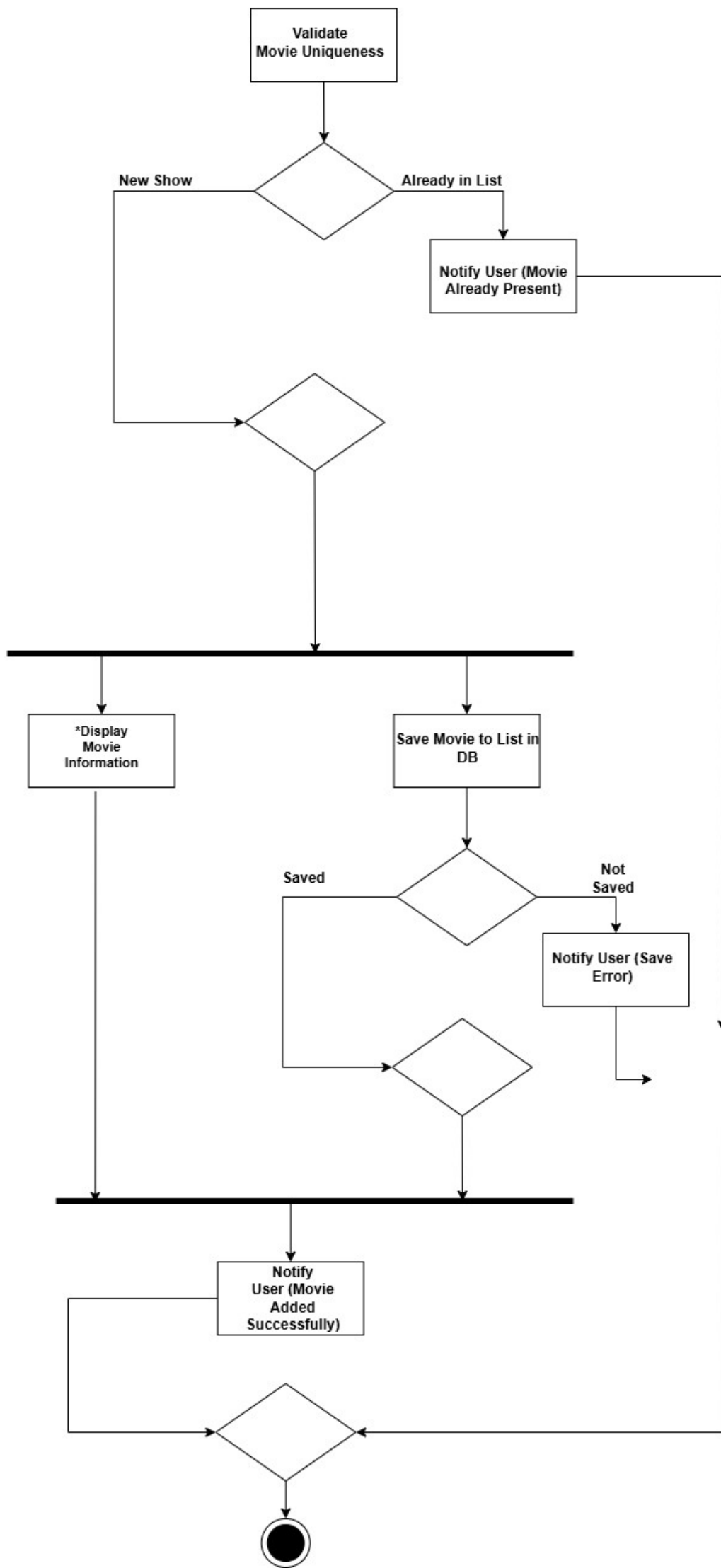
In the context of the system, the **Facade** pattern has been implemented, primarily through the **MovieTmdb** class. The main point of this pattern is to provide a simplified, high-level interface to a complex subsystem.

The MovieTmdb class serves as a facade for the TMDB API interaction. This design was a significant help because it abstracts away the complexities of making HTTP requests using OkHttpClient, handling API keys, parsing JSON responses with Gson, and managing various error conditions. Consequently, client code (e.g., other parts of your application that need movie information) does not need to know the intricate details of the TMDB API's structure, the specifics of HTTP communication, or the mechanics of JSON deserialization. Instead, they can rely on the simplified interface provided by MovieTmdb (e.g., getMovieById, searchMovies), simplifying their design and ensuring a clean separation of concerns.

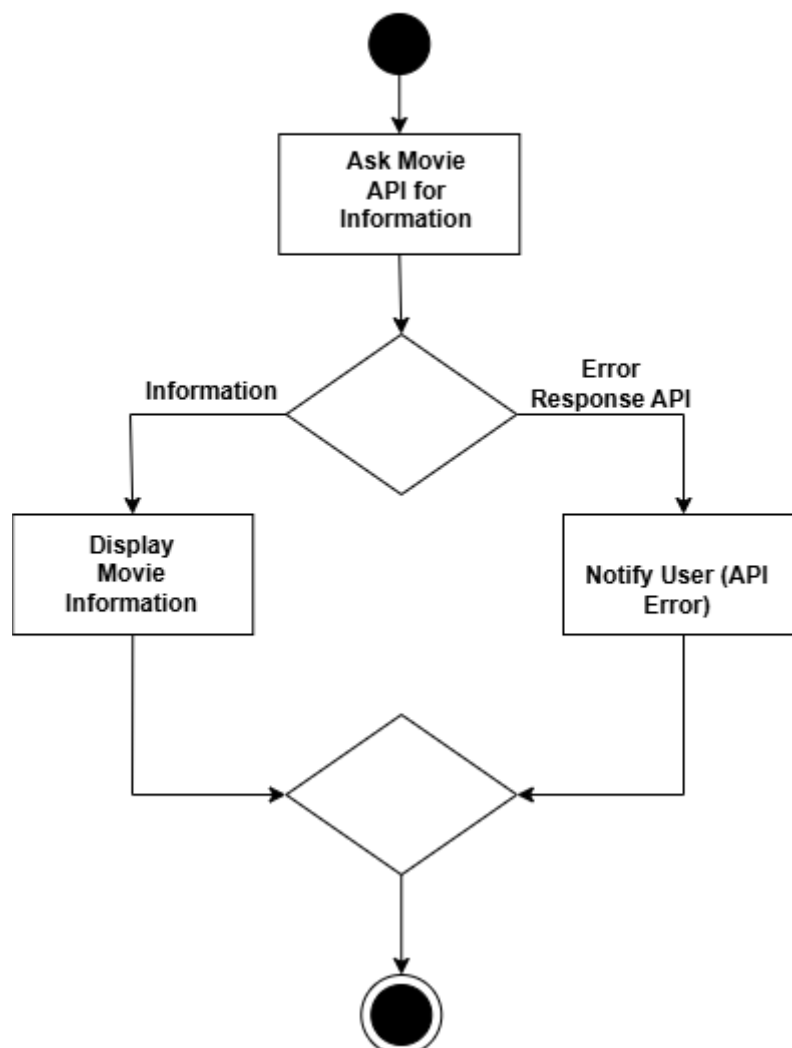
### 3.3. ACTIVITY DIAGRAM

## Add Movie to Personal List



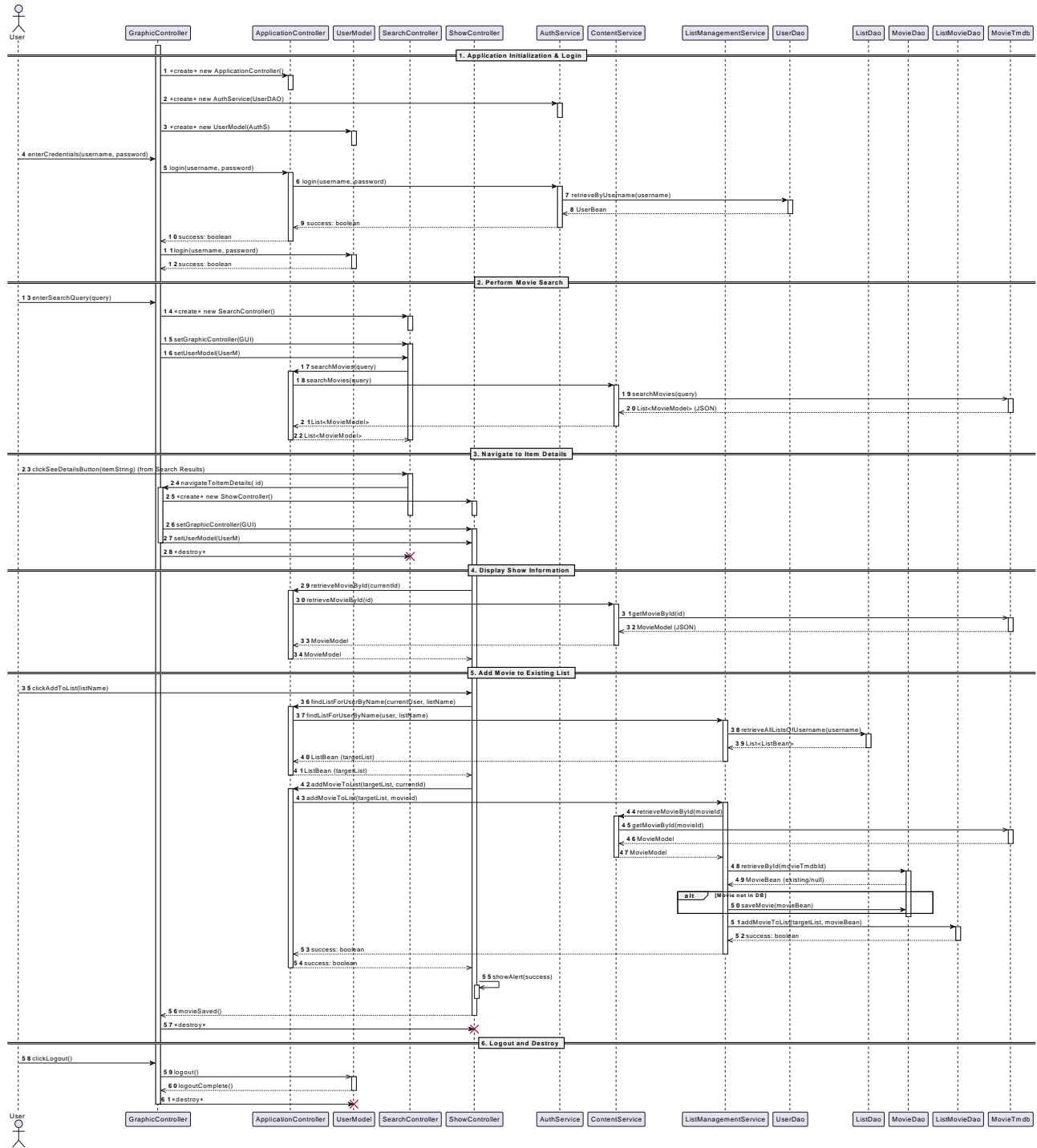


## Display Movie Information



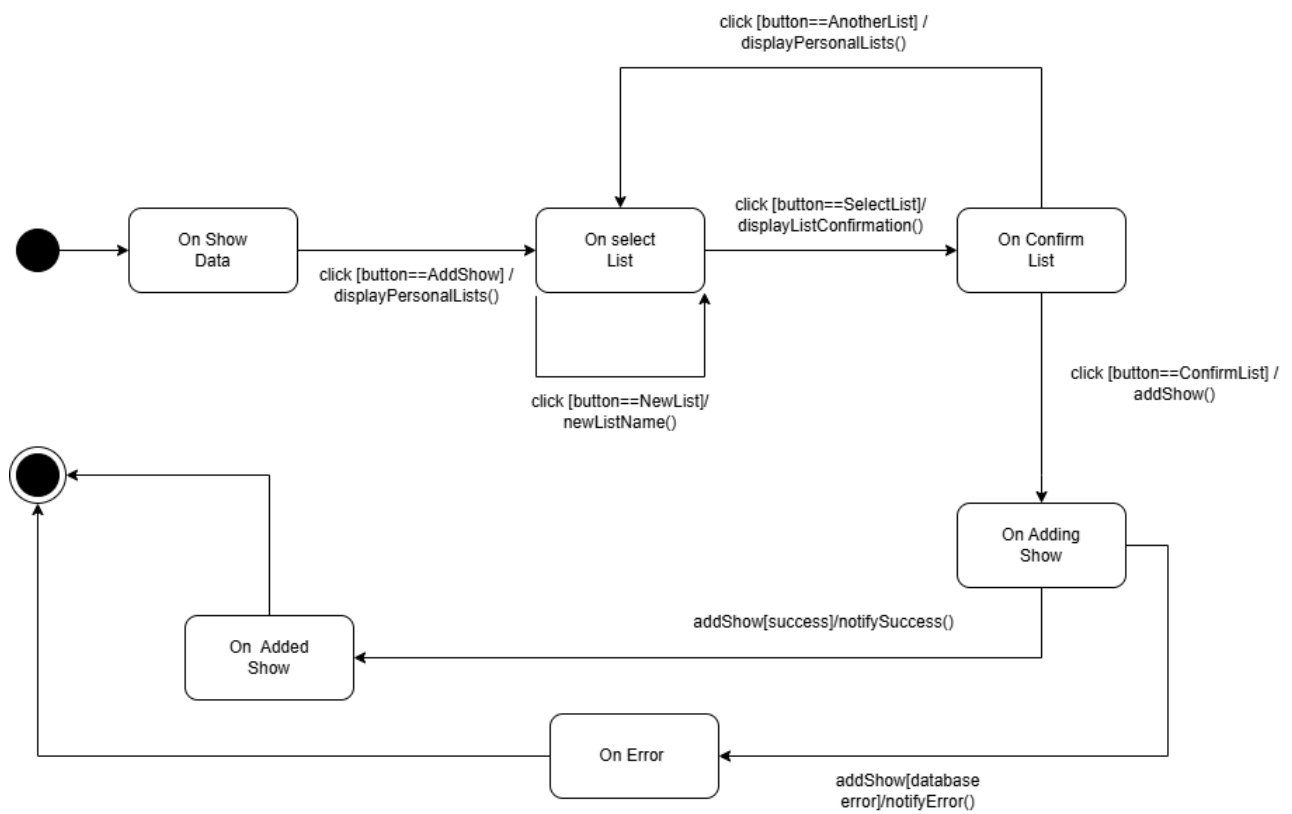
### 3.4. SEQUENCE DIAGRAM

This diagram is attached near to this document to be able to have a closer look to it.





### 3.5. STATE DIAGRAM



## 4. TESTING

In the testing part, we have three test classes in which some of the most important or relevant functions of the classes are tested.

In first place, we have the **TestListDaoInMemory**, that oversees testing the ListDaoInMemory class and the functions: saveList(), retrieveById(), deleteList(), and retrieveAllListsOfUsername().

In second place, we have the **TestMovieDaoInMemory**, that oversees testing the MovieDaoInMemory class and the functions: retrieveById(), saveMovie(), and retrieveAllMovies().

In third and last place, we have the **TestUserDaoInMemory**, that oversees testing the UserDaoInMemory class and the functions: saveUser() and retrieveByUsername().

## 5. EXCEPTIONS

In handling exceptions, the system utilizes dedicated custom exception classes to manage specific error scenarios:

1. **ExceptionDatabase:** This class inherits from RuntimeException and is responsible for handling exceptions related to database operations. It is used in SingletonDatabase to oversee issues such as:
  - Failure to locate or load the database.properties file.
  - Missing essential database configuration properties (URL, USER, PASSWORD).
  - Inability to find the MySQL JDBC Driver.
  - Failures during the establishment or re-establishment of a database connection.
  - Problems encountered when attempting to gracefully close a database connection.
2. **ExceptionTmdbApi:** Also inheriting from RuntimeException, this class manages exceptions that occur during interactions with the TMDB API. It is used by MovieTmdb to handle situations including:
  - The tmdbapi.properties file not being found, or the TMDB\_API\_KEY being absent or empty.
  - Network errors or non-successful HTTP responses during API requests.
  - Errors in parsing JSON responses received from the TMDB API.

## **6. DATA BASES**

### **6.1. FILE SYSTEM**

The file system is a set of files with the information of the entities. This file distribution is the following:

1. fileData/csvData/user.csv: This CSV file contains information related to users.
2. fileData/csvData/list.csv: This CSV file contains list information.
3. fileData/csvData/movie.csv: This CSV file stores data about movie entries.
4. fileData/csvData/listmovie.csv: This CSV file contains data about the user list and the movies in that list.

### **6.2. MYSQL**

The other type of DBMS in the system is the connection to a MySQL server to access the information. Internally this SQL must follow the struct specified in the file Project\_ISPW.sql.

## **7. SONAR CLOUD**

To check the correct behavior of this project, it was used SonarCloud system. This system points out the possible bugs, possible vulnerabilities, and review that the functions are well implemented. You can check the results of the project in the following link:

[https://sonarcloud.io/summary/overall?id=IlieManoliUni\\_Project\\_ISPW&branch=master](https://sonarcloud.io/summary/overall?id=IlieManoliUni_Project_ISPW&branch=master)