

## Table of Contents

Design Document.....	2
1. System Overview.....	2
• Brief Project Description: .....	2
• System Architecture:.....	2
• Technology Stack: .....	3
2. Implementation Details .....	3
• Codebase Structure: .....	3
• Key Implementations: .....	4
• Component Interfaces: .....	5
• Visual Interfaces: .....	6
3. Use Case Support in Design.....	7
• Use Case Selection: .....	7
• Requirement Mapping: .....	8
• Use Case Design: .....	8
• Demo Requirement: .....	10
4. Design Decisions .....	11
• Technology Comparisons: .....	11
• Decision Justifications: .....	11
5. Github Commit Requirement .....	12
• Code Implementations & Interface: .....	12
• Technology Comparisons: .....	12
Task Matrix .....	12

# Design Document

Esma Nur Akçınar

Alime Feyza Şahin

Merve İçkilli

Tarık Şamil Şahin

## 1. System Overview

- **Brief Project Description:** MovieNest is a desktop application that allows users to search for movies and series, filter them by genre, create watchlists, and receive recommendations. The platform integrates TMDb API for fetching movie data and IMDb ratings.

The aim of the platform is to help users search for movies with its understandable interface and ease of use, and to allow them to keep a list where they can store the movies they want to watch.

- **System Architecture:** Our desktop app will follow a layered architecture. The architecture can be divided into the following layers:

### *Presentation Layer (UI Layer) – Qt (PyQt)*

- Responsible for displaying the user interface.
- Handles user interactions like searching, filtering, and managing the watchlist.

### *Application Layer (Business Logic) – Python*

- Manages the core logic, such as fetching data from the TMDb API, applying filters, and handling recommendations.
- Processes user requests and communicates between the UI and data layers.

### *Data Layer – MySQL & TMDb API*

- Stores user preferences, watchlists, and cached movie data in MySQL.
- Fetches movie details from the TMDb API dynamically when needed.

### *External Services – TMDb API*

- Acts as a third-party service providing movie details, genres, ratings, and other metadata.

This architecture keeps the system modular and scalable.

- **Technology Stack:** Our project will use the following technologies:

#### *Front-End (UI Layer)*

- Qt (PyQt) – For designing the desktop UI.
- Qt Designer – For visually designing UI components.

#### *Back-End (Business Logic Layer)*

- Python – Main programming language for logic implementation.
- Requests – For making API calls to TMDB.

#### *Database (Data Layer)*

- MySQL – For storing user data like preferences, watchlists, and cached movies.

#### *External API*

- TMDB API – For fetching movie details, genres, and ratings.

#### *Other Tools*

- VS Code – For coding.
- Git/GitHub – For version control
- Notion – For task management

## 2. Implementation Details

- **Codebase Structure:**

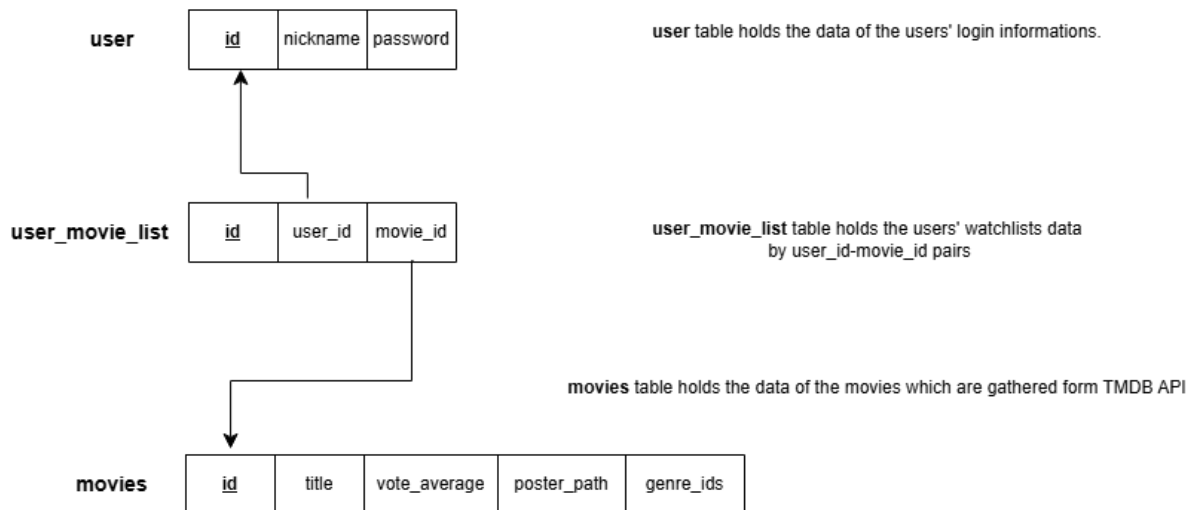
movie desktop app/

- main.py (Entry point, initializes application)
- movie\_database.py (Handles MySQL and TMDB API connections)
- ui\_main\_page.py (Main page GUI: search bar, categories, recommendations)
- ui\_login\_page.py (User login interface)
- ui\_watchlist\_page.py (Watchlist page, displays saved movies)
- resources/ (Static assets: icons, images, etc.)
- README.md (Project documentation)

- **Key Implementations:**

- *UI Layer (` `)*: Manages user interface (PyQt windows for searching, displaying details, managing watchlists).
- *Application Layer (` `)*: Implements core business logic like search, filtering, and watchlist operations.
- *Data Layer (` `)*: Handles database interactions for persisting user watchlists. Uses a **local MySQL database** to store user profiles, movie details, and watchlists.
- *Relational table of the database:*

## movienest



Search API: Fetches movies from TMDB API.

Watchlist Management: Allows users to add/remove movies locally.

Recommendation Algorithm: Suggest movies based on IMDb rating.

Filtering System: Enables filtering by genre and rating.

Sorting System: Enables sorting by name and rating.

- **Component Interfaces:**

`login_user(username, password)`: Checks the database for the given username-password pair and authenticates the user.

`register_user(username, password)`: Checks the database for the given username-password pair and creates a user.

`filter(genre[ ], rating[ ]) return [ ][ ]`: Makes a filtering operation on the movie list based on the genre and rating filtering options. Returns the filtered movie list.

`search_and_filter(genre[ ], rating[ ]) return [ ][ ]`: Makes a filtering operation on the movie list based on the genre and rating filtering options. Returns the filtered movie list.

`search(title) return [ ][ ]`: Searches the given title in the movie list and returns the results.

`sort(sort_type, order, flag) return [ ][ ]`: Makes a sorting operation by sort type and sort order on the given list which is specified by a flag parameter.

`get_watchlist(user_id) return [ ][ ]`: Returns the watchlist of the user.

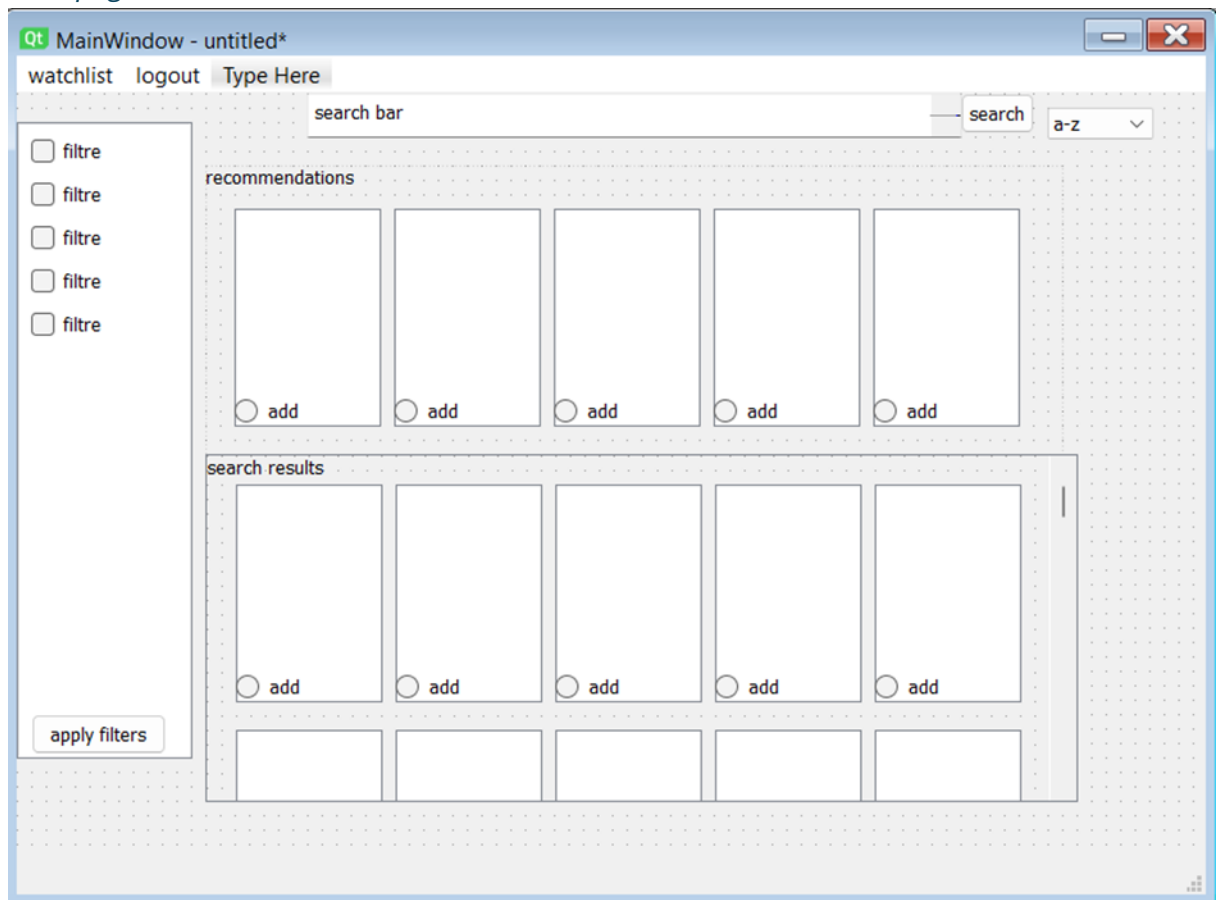
`add_movie(movie_id, user_id) return true`: Adds the specified movie to the user's watchlist.

`remove_movie(movie_id, user_id) return true`: Removes the specified movie from the specified user's watchlist.

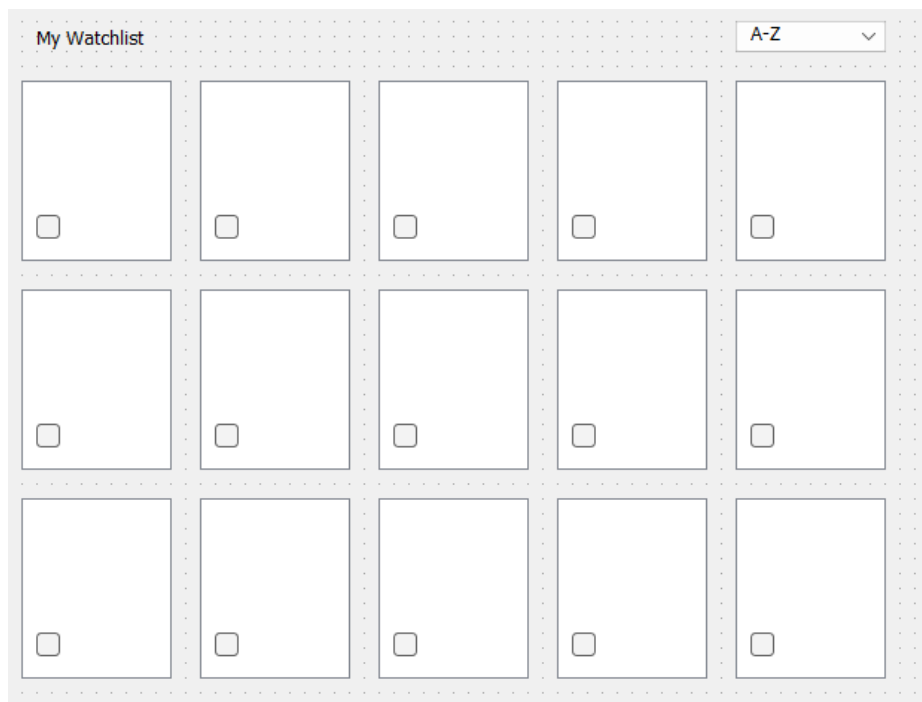
`recommend_movie() return [ ][ ]`: Returns a movie list based on ratings.

- Visual Interfaces:

- *Main page*

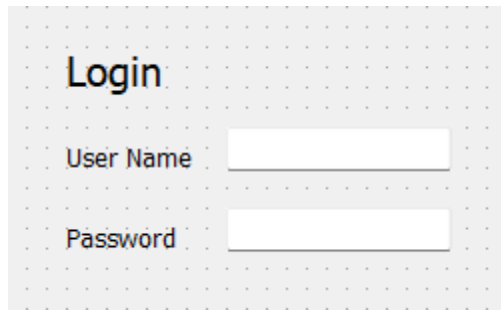


- *Watchlist Page*



-

- *Login Page*

A mockup of a login form on a light gray background with a subtle dot pattern. The form is centered and contains the following elements: the word "Login" in a bold, black, sans-serif font; a label "User Name" in a smaller, black, sans-serif font followed by a white rectangular input field; and a label "Password" in a smaller, black, sans-serif font followed by a white rectangular input field.

Login

User Name

Password

- 

### 3. Use Case Support in Design

- Use Case Selection:
- Searching for movies and series: Users can search by name using a search bar.
- Filtering search results by genre: Clicking a category button filters results.
- Sorting search results alphabetically or by ratings: Users can sort the movie lists by selecting a sorting method.
- Managing watchlist: Users can add/remove movies from their watchlist.

- **Requirement Mapping:**

1. Use case: Searching for a movie.

Functional Requirement: Users can create profiles. Users can access their accounts by username and password. Users must be able to search movies and series by titles.

2. Use case: Filtering by category

Functional Requirement: User could filter movie list based on selected filters.

The platform must support efficient search functionality with filtering options.

4. Use case: Sorting search results

Functional Requirement: Users could sort search results based on alphabetic order and ratings.

Users could sort their watchlist based on alphabetic order and ratings.

5. Use case: Manage watchlist

Functional Requirement: Users must be able to create and manage a watchlist.

- **Use Case Design:** Each use case is supported by modularized components in the frontend (PyQt UI), business logic (Python classes and methods), and data layer (MySQL + TMDB API).

*Use case 1:*

1. The user opens the application and the login screen occurs.

○ If Login option selected:

The user enters their username and password into the input fields.

The system validates whether both fields are filled in correctly.

The system sends the credentials to the database for verification.

The database checks if the password matches the stored hash.

If authentication is successful, the user is granted access to the main application.

If authentication fails, an error message is displayed, prompting the user to re-enter credentials.

○ If Sign-up option selected:

The user enters their username and password into the input fields.

The system validates the input (username availability).

The system hashes the password and stores the new user credentials in the database.



The user is granted access to the main application after successful account creation.

2. The user navigates to the main page after a successful login.
3. The user navigates to the movie search section.
4. The user types a movie title (or part of a title) into the search bar.
5. The system sends a query to the database, searching for matching movie titles.
6. The database returns a list of movies matching the search criteria.
7. The system displays the search results in the UI.

System Architecture Support:

Frontend (PyQt5 UI): Handles username and password input and displays results. Handles search input and displays results.

Backend (Python Logic): Sends SQL queries to retrieve username-password pairs and matching movies. Uses authorization function to manage user login.

Database (MySQL): Stores user and movie data and executes queries to fetch results.

#### *Use case 2:*

1. The user navigates to the main page after a successful login.
2. The user selects a category (e.g., Action, Comedy, Drama) or a specified rating range from the filter options.
3. The system updates the displayed movie list based on the selected category.
4. The filtered movies are retrieved from the database using a query filtering by the selected genre and filter options.
5. The UI updates results to show only movies that match the selected category.

System Architecture Support:

Frontend (PyQt5 UI): Provides filter buttons/dropdowns for users to select categories.

Backend (Python Logic): Generates SQL queries based on selected filters.

Database (MySQL): Executes queries to fetch only movies matching the selected filters.

#### Use case 3:

1. The user navigates to the main page or watchlist page after a successful login.
2. The user selects a sorting option (e.g., Sort by Title, Sort by Rating) from the sorting dropdown menu.
3. The system applies the selected sorting criterion to the displayed list of movies using sorting algorithms. If sorting search results, the system reorders movies on the main page. If sorting watchlist, the system reorders movies within the user's saved list.
4. The UI updates the displayed movie list to show movies in the desired order.
5. If the user selects another sorting criterion, the process repeats with the new criterion.

System Architecture Support:

Frontend (PyQt5 UI): Dropdown UI elements allow sorting selection.

Backend (Python Logic): Calls sorting functions for search results or watchlist data. Sorts the fetched movie lists from the database using backend's sorting functions.

Database (MySQL): Executes queries to fetch movies.

#### Use case 4:

1. The user navigates to the main page after a successful login.
2. The user clicks the "Add to Watchlist" button located under the desired movie.
3. The system checks if the movie is already in the user's watchlist.
4. If it's already in the watchlist, the system gives a pop-up warning to the user.
5. If not, the system sends a request to the MySQL database to add the movie under the user's profile.
6. The database stores the movie in the user's personal watchlist.
7. The user can navigate to their Watchlist section to see saved movies.
8. The user can remove movies from the watchlist, which updates the database accordingly.

System Architecture Support:

Frontend (PyQt5 UI): Displays an "Add to Watchlist" button and updates the UI when movies are added/removed.

Backend (Python Logic): Sends database requests to store or delete a movie from the user's watchlist.

Database (MySQL): Inserts/removes watchlist records based on user actions

- **Demo Requirement:**

- Searching for a movie
- Filtering by category
- Sorting watchlist and results

- Managing watchlist

## 4. Design Decisions

- Technology Comparisons:

*Programming Language: Python vs. C++/Java*

Python was chosen over C++ and Java due to its simplicity, extensive libraries, and rapid development capabilities. Unlike Java, which requires a more complex setup, and C++, which involves manual memory management, Python allows for faster prototyping and integration with various tools.

*Database: MySQL vs. SQLite vs. PostgreSQL*

We opted for MySQL instead of SQLite and PostgreSQL. SQLite is a lightweight option but lacks the scalability needed for potential future extensions. PostgreSQL is highly robust but more complex to set up and manage. MySQL provides an optimal balance between performance, reliability, and ease of use, making it ideal for a desktop application that requires structured data storage.

*GUI Development: PyQt (Qt Designer) vs. Tkinter vs. Kivy*

The GUI was developed using PyQt with Qt Designer, as it offers a more modern and visually appealing interface compared to Tkinter. While Kivy is a strong alternative for cross-platform applications, PyQt provides a more professional look and integrates seamlessly with the Python ecosystem. Qt Designer further simplifies UI design.

- Decision Justifications:

The following justifications support our technology choices:

*Python:*

- High readability and ease of maintenance.
- Large ecosystem with libraries for database interaction, networking, and GUI development.
- Faster development time compared to low-level languages like C++.

*MySQL:*

- Supports structured queries and transactions, ensuring data integrity.
- More scalable than SQLite, allowing potential expansion of the application.
- Widely used and well-documented, making troubleshooting easier.

*PyQt & Qt Designer:*

- Provides a feature-rich UI framework with cross-platform support.
- More visually appealing and flexible than Tkinter.
- Qt Designer enables rapid UI prototyping, improving development efficiency.

By combining these technologies, we ensure a balance between performance, usability, and scalability while maintaining a structured and maintainable codebase.

## 5. Github Commit Requirement

- Code Implementations & Interface:
- Technology Comparisons:

### Task Matrix

Tasks	Esma Nur	Feyza	Merve	Tarık Şamil
System Overview	. <input checked="" type="checkbox"/>	. <input checked="" type="checkbox"/>	. <input checked="" type="checkbox"/>	. <input checked="" type="checkbox"/>
Implementation Details	. <input type="checkbox"/>	. <input checked="" type="checkbox"/>	. <input type="checkbox"/>	. <input type="checkbox"/>
Use Case Support in Design	. <input checked="" type="checkbox"/>	. <input type="checkbox"/>	. <input checked="" type="checkbox"/>	. <input checked="" type="checkbox"/>
Design Decisions	. <input checked="" type="checkbox"/>	. <input checked="" type="checkbox"/>	. <input checked="" type="checkbox"/>	. <input type="checkbox"/>
Github Commit Requirement	. <input type="checkbox"/>	. <input type="checkbox"/>	. <input type="checkbox"/>	. <input checked="" type="checkbox"/>