

# Movies Watchlist

(SVVT Documentation)

June, 2025.



<b>Introduction.....</b>	<b>4</b>
About the Project.....	4
Project Functionalities and Screenshots.....	5
Landing Page.....	5
Add Movie Functionality.....	6
Edit Movie Functionality.....	7
Delete Movie Functionality.....	8
Mark Movie as Watched Functionality.....	9
Add New Category Functionality.....	10
Edit Category Name Functionality.....	10
Delete Category Functionality.....	11
Filtering Functionality.....	12
Sorting Functionality.....	13
Change Notifications Status Functionality.....	14
<b>Test Plan.....</b>	<b>15</b>
Purpose and Scope.....	15
Testing Types and Strategy.....	15
Unit Testing (Backend).....	15
Integration Testing.....	16
System Testing (Selenium).....	16
Static Code Analysis.....	17
Code Coverage Analysis.....	17
Entry and Exit Criteria.....	17
Testing Environment and Tools.....	18
Deliverables.....	20
<b>Test Execution.....</b>	<b>20</b>
Unit Testing.....	20
Integration Testing.....	22
System Testing (Selenium).....	25
Test Case 1 - Add a movie with valid data.....	25
Test Case 2 - Edit an existing movie.....	26
Test Case 3 - Delete a Movie.....	28
Test Case 4 - Mark Movie as Watched.....	29
Test Case 5 - Apply Filters.....	31
Test Case 6 - Apply Sorting.....	32
Test case 7 - Add New Category.....	33
Test Case 8 - Edit Category.....	35
Test Case 9 - Delete Category.....	36



Static Code Analysis.....	38
Code Coverage Analysis.....	40
<b>Conclusion.....</b>	<b>41</b>
Entry and Exit Criteria Overview.....	41
Deliverables Summary.....	42
Project Summary.....	42



## Introduction

### About the Project

Movies Watchlist is a web-based application designed to help users efficiently manage their personal movie watchlists. It allows users to add, edit, filter, and mark movies as "Watched" while leveraging AI-driven genre suggestions and email notifications with personalized movie recommendations. The application emphasizes easy usage, responsiveness, and scalability.

### Core Application's Features are:

- Add, edit, and delete movies
- AI-based genre suggestions via OpenAI API
- Mark movies as "Watched" and receive email notifications with similar movie recommendations
- Filtering and sorting by genre, status, watchlist priority, and category
- Manage custom watchlist categories (e.g., "Watch after exams", "During spring break", etc.)
- Enable or disable email notifications
- Responsive user interface for all device types

### Application's Technology Stack:

- Frontend: React
- Backend: Java Spring Boot
- Database: MySQL
- External APIs: OpenAI API (AI recommendations), Infobip API (email delivery)

**Live Application Link:** <https://movie-watchlist-fe-live.onrender.com/>

**Please note:** If the deployed application link appears to be unavailable, this is likely due to Render's free-tier service entering sleep mode after a period of inactivity. If you encounter this issue, notify me and I will immediately restart the deployment.



## GitHub Repositories:

- Backend Code, Unit and Integration Tests: [Movie-Watchlist-BE-Spring-SE-Project](#)
- Frontend: [Movies-Watchlist-FE-React-SE-Project](#)
- Selenium UI tests: [SVVT Master](#)

## Project Functionalities and Screenshots

This section outlines the core functionalities of the Movies Watchlist application, describing how users interact with its interface to manage and organize their movie watchlists effectively. The application emphasizes simplicity, clarity, and responsiveness across devices.

### Landing Page

Upon logging in, users are taken to the **My Movies** dashboard — the central hub of the application where all core features are accessible from a single, intuitive interface.

Key elements available on this page include:

- **Sorting dropdown** to organize movies alphabetically or by custom-defined watchlist order.
- **Multiple filters** for narrowing the list by genre, status (e.g., Watched/To Watch), or watchlist order.
- **Watchlist category selector** to display only movies from a specific custom group.
- **Buttons for adding, editing, or deleting categories**, enabling dynamic grouping of movies.
- **Global notification toggle** that allows users to turn email notifications on or off.
- A button to **open the Add Movie modal**, where new entries can be submitted manually or with AI assistance.

This page also displays each movie as a neatly formatted card showing:

- Title, genre, description, and status
- Watchlist order and associated categories
- Action icons for marking as watched, editing, or deleting

## MY MOVIES

Sort Movies
Filter by Genre
Filter by Status
Filter by Watchlist Order
CHANGE YOUR NOTIFICATION STATUS
ADD MOVIE MODAL

Sort Movies
Filter by Genre
Filter by Status
Filter by Watchlist Order
APPLY FILTERS
RESET FILTERS

**Watchlist Category Option**

Select Category
ADD NEW CATEGORY
EDIT CATEGORY
DELETE CATEGORY

**Focus**

**Genre:** Action

**Description:** Rewatch during break

**Status:** Watched

**Watch Order:** When I have time

**Categories:** Movies To Wait For, When exams end

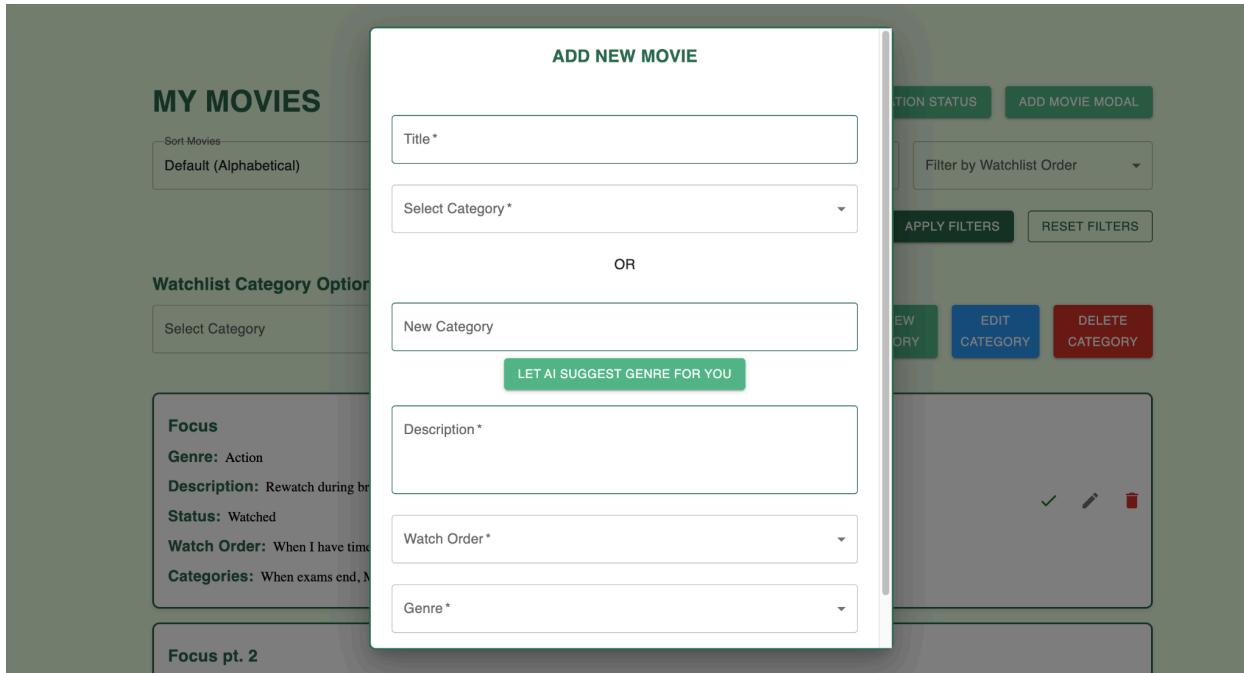
✓ ✎ ✖

## Add Movie Functionality

When a user clicks the "**Add Movie Modal**" button, a dynamic form appears. Key fields include:

- **Title** (required)
- **Description** (required)
- **Category** (select existing or create new)
- **Watchlist Order** (e.g., "Someday", "Next Up")
- **Genre** (manual or AI-suggested)

Clicking the "**Let AI suggest genre for you**" button uses OpenAI's API to assign the most suitable genre based on the movie's title and description.



## Edit Movie Functionality

Each movie in the user's watchlist can be edited at any time. By clicking the pencil icon on a movie card, a modal titled "**Edit Movie**" appears. It displays all existing information in editable fields:

- **Title and Description**
- **Genre**, which can be updated from the predefined list
- **Watchlist Order** (e.g., "Next Up", "When I have time")
- **Watchlist Categories**, where multiple categories can be selected
- An optional field to create and assign a **new category** on the fly

This feature allows users to refine or reorganize their movie entries as their preferences change. Upon saving changes, the updated movie details are instantly reflected on the dashboard.



The screenshot shows a user interface for managing a movie watchlist. On the left, there's a sidebar titled "Watchlist Category Option" with a dropdown menu set to "Select Category". Below it are two movie entries: "Focus" and "Focus pt. 2". Each entry includes fields for Title, Description, Genre, Status, Watch Order, and Categories. The "Focus" entry has "Action" as its genre and "Rewatch during break" as its description. The "Focus pt. 2" entry has "Thriller" as its genre and "Demo" as its description. On the right, a central panel titled "EDIT MOVIE" contains fields for "Title" (set to "Focus"), "Description" (set to "Rewatch during break"), "Genre" (set to "Action"), "Watch Order" (set to "When I have time"), and "Categories" (set to "When exams end, Waiting Movie Premiere"). Below these fields is an "OR" button and a "New Category" input field. At the top right of this panel are "APPLY FILTERS" and "RESET FILTERS" buttons, and below them are three buttons: "NEW", "EDIT CATEGORY" (highlighted in blue), and "DELETE CATEGORY". To the right of the main panel are two movie cards for "Focus" and "Focus pt. 2", each with a green checkmark, a pencil icon, and a red trash icon.

## Delete Movie Functionality

The red trash icon on each movie card allows users to initiate the delete operation. When clicked, a confirmation dialog appears to ensure the user intends to remove the selected movie. It includes the movie's title and a warning that the action is irreversible.

This safety confirmation helps prevent accidental deletions. Once confirmed, the movie is permanently removed from both the database and the visual watchlist.

The screenshot shows the same interface as the previous one, but with a modal dialog box centered over the "Focus" movie card. The dialog is titled "DELETE MOVIE" and contains the message "Are you sure you want to delete "Focus"? This action cannot be undone." At the bottom of the dialog are two buttons: "CANCEL" (in blue) and "DELETE" (in red). The background of the main interface is dimmed to indicate that interaction with it is disabled while the dialog is open. The sidebar and other movie cards are visible but inactive.



## Mark Movie as Watched Functionality

Users can track their viewing progress by marking a movie as "**Watched**." This functionality is accessible directly from the movie card, using a checkmark icon. Once clicked, the movie's status is updated both visually on the frontend and persistently in the backend database.

In addition to updating the status, the system performs the following:

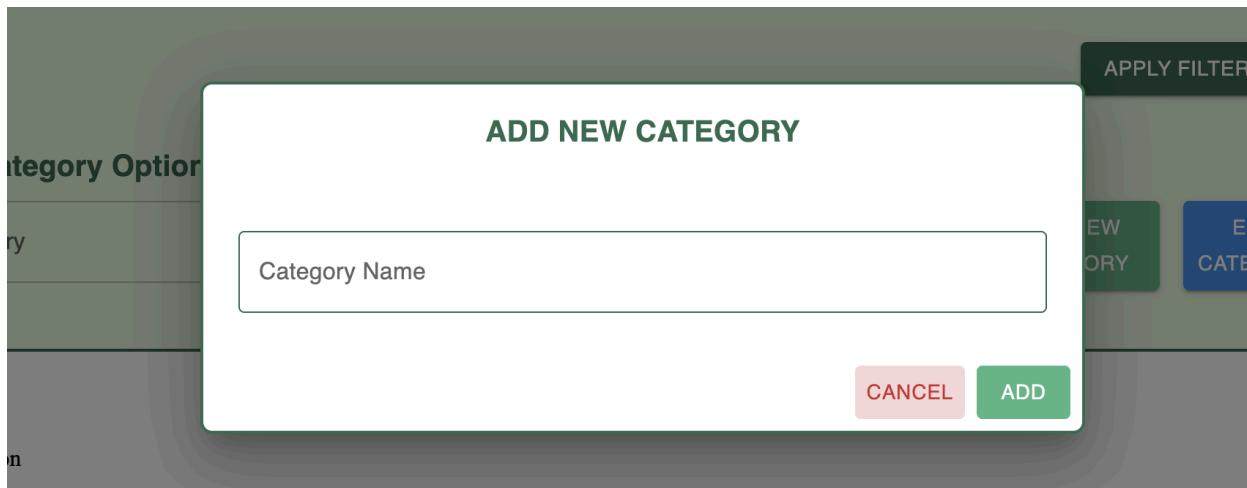
- **Triggers an email notification** (if the user has email notifications enabled)
- The email includes a confirmation message and **five AI-generated recommendations** for similar movies from the same genre, powered by the OpenAI API
- The movie remains visible in the user's watchlist but is now clearly labeled as "Watched," allowing users to revisit or reference it in the future

This feature supports user motivation and organization by providing both functional tracking and intelligent recommendations for continued engagement.

A screenshot of a movie watchlist interface. The interface shows three movie cards: "Genre: Action", "Description: Rewatch during break", "Status: Watched", "Watch Order: When I have time", and "Categories: When exams end, Waiting Movie Premiere"; "Focus pt. 2", "Genre: Thriller", "Description: Demo", "Status: Watched", "Watch Order: When I have time", and "Categories: When exams end"; and "Mission Impossible", "Genre: Action", "Description: Action movie", "Status: To Watch", "Watch Order: Next Up", and "Categories: When exams end". A central modal dialog box titled "MARK AS WATCHED" contains the text "Are you sure you want to mark \"Mission Impossible\" as watched? This action cannot be undone." with "CANCEL" and "CONFIRM" buttons. The background cards have edit and delete icons in the top right corner.

## Add New Category Functionality

Users can create custom groupings (e.g., “Watch During Holidays”, “To Watch with Friends”) and assign movies accordingly.



## Edit Category Name Functionality

Users can rename existing categories through a clean interface. The new name reflects across all associated movies, preserving organization without data loss.



Filter by Genre

Filter by Status

APPLY

CREATE

EDIT

DELETE

SEARCH

Logout

### EDIT CATEGORY

Select Category

Movies To Wait For

New Category Name

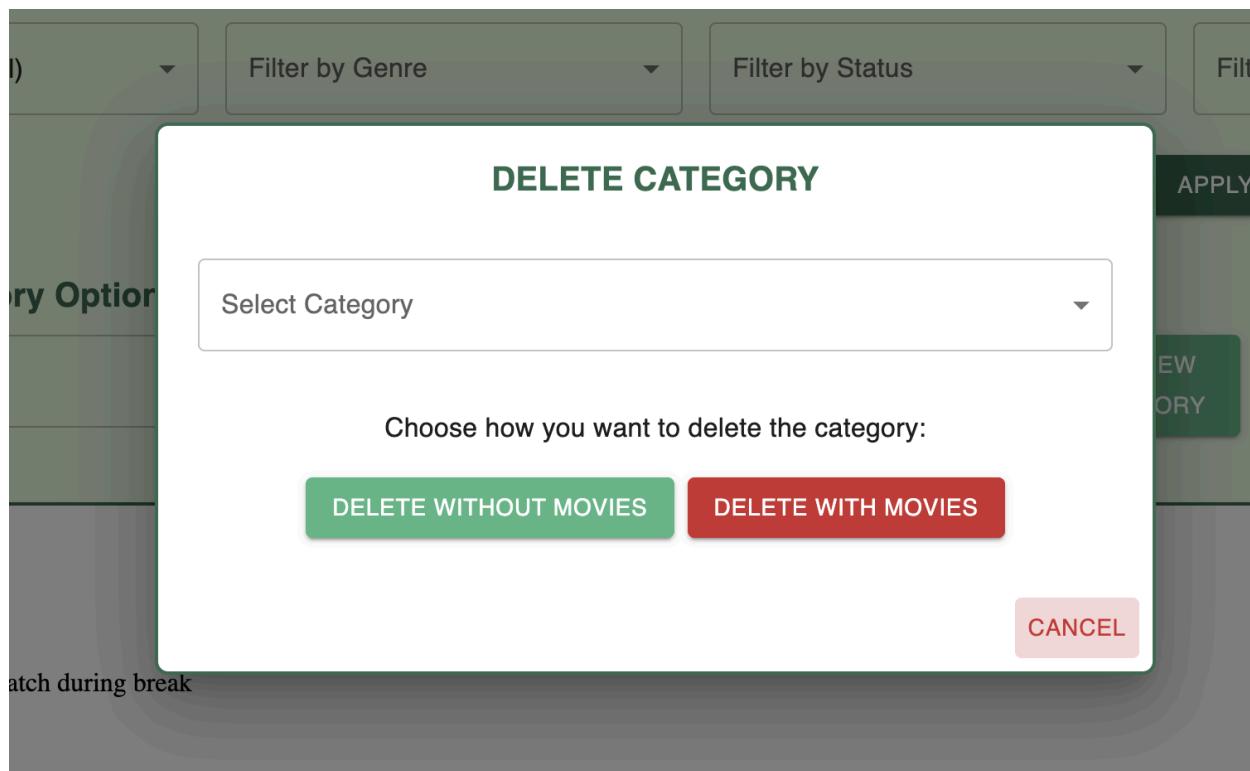
CANCEL

SAVE CHANGES

## Delete Category Functionality

To give users users flexible control over their watchlist structure, two options are provided when deleting:

- **Delete without movies** – keeps all associated movies but unassigns their category.
- **Delete with movies** – permanently removes both the category and the movies inside.



## Filtering Functionality

To enhance navigation and movie discovery, the dashboard provides multiple filtering options through dropdowns:

- **Genre** (e.g., Action, Comedy, Thriller)
- **Status** (To Watch / Watched)
- **Watchlist Order** (e.g., “Next Up”, “Someday”)
- **Watchlist Category Option** — a custom grouping feature allowing users to narrow results by user-defined categories (e.g., “When exams end”, “Ski trip movies”)

These filters can be applied in combination using the “**Apply Filters**” button. The “**Reset Filters**” option clears all selections and returns the full watchlist view.

## MY MOVIES

CHANGE YOUR NOTIFICATION STATUS    ADD MOVIE MODAL

Sort Movies    Filter by Genre    Filter by Status    Filter by Watchlist Order

APPLY FILTERS    RESET FILTERS

**Watchlist Category Option**

Select Category

None  
Waiting Movie Premiere  
When exams end

Description: Rerun during break    ✓    Edit    Delete

Status: Watched

Watch Order: When I have time

Categories: When exams end, Waiting Movie Premiere

## Sorting Functionality

Movies are sorted alphabetically by title by default. Users can also sort their watchlist based on the watch order they assigned (ascending or descending), enabling personalized prioritization of what to watch next.

## MY MOVIES

CHANGE YOUR NOTIFICATION STATUS    ADD MOVIE MODAL

Sort Movies    Watchlist Order (Asc)    Filter by Genre    Filter by Status    Filter by Watchlist Order

APPLY FILTERS    RESET FILTERS

**Watchlist Category Option**

Select Category

**Mission Impossible**

Genre: Action  
Description: Action movie  
Status: Watched  
Watch Order: Next Up  
Categories: When exams end

✓    Edit    Delete

**Focus**

## Change Notifications Status Functionality

The application includes a toggle for enabling or disabling **email notifications**. This functionality is accessible through the “**Change Your Notification Status**” button located in the top right section of the dashboard.

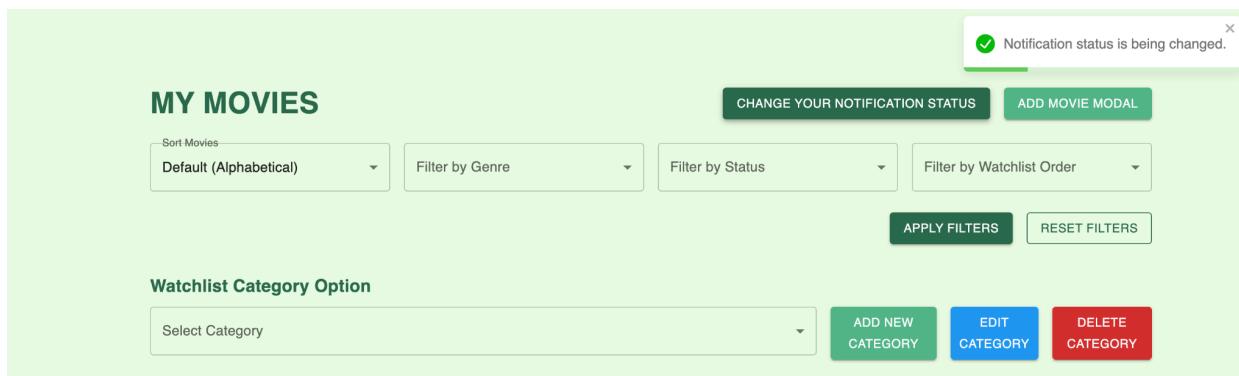
When enabled:

- The system sends an email to the user every time a movie is marked as “Watched”
- The email contains both a confirmation and five AI-recommended movie suggestions from the same genre, powered by OpenAI

When disabled:

- No emails are sent, allowing users to manage their preferences based on need, context, or privacy concerns

Upon clicking the button, a real-time alert appears confirming the notification preference has been updated.





## Test Plan

### Purpose and Scope

The purpose of this test plan is to define the testing strategy, tools, and activities used to verify and validate the functionality, quality, and reliability of the **Movies Watchlist** application. This includes unit testing, integration testing, system testing (UI), static code analysis, and code coverage reporting. The goal is to ensure the application meets its functional requirements as outlined in the chapter above.

Within the scope of this test plan are all major functionalities provided by the application, including:

- Adding, editing, and deleting movies
- Marking a movie as watched and sending a recommendation email
- Filtering and sorting movies by genre, watchlist order, and status
- Managing user-defined watchlist categories (create/edit/delete)
- Enabling or disabling email notifications

### Testing Types and Strategy

The testing approach includes unit testing, integration testing, system (UI) testing, and static analysis of the source code.

#### Unit Testing (Backend)

Unit testing will be implemented using **JUnit 5**, and it will target key backend layers of the application: the **DAO (repository) layer**, **service layer**, and **controller layer**. These tests are designed to verify that individual components behave as expected in isolation, without relying on external systems.

The repository layer will use mocked return values (not the actual database information) to confirm proper data retrieval and storage operations. Mocked values keep the database integrity intact. The service layer contains the core business logic and it will include tests for operations such as adding movies, marking them as watched, editing attributes, and handling invalid inputs.



Controller tests will validate the behavior of REST endpoints using mocked service responses and confirm proper HTTP status codes and response structures.

All test methods will be isolated from the database using **Mockito library**, which ensures data consistency during testing and allows precise control over test scenarios. The overall goal is to validate internal application behavior at the method level before progressing to integration and system-level tests.

## Integration Testing

Integration testing will focus on verifying the correct interaction and data flow between multiple layers and components of the backend application. In the context of the Movie Watchlist system, this primarily involves the coordination between:

- **Controllers and Service Layers**
- **Service Layer and Repository/Data Access Layer**

The goal is to ensure that when components work together, they behave as expected and consistently fulfill the business requirements.

## System Testing (Selenium)

System tests simulate real-world usage scenarios through the user interface using **Selenium WebDriver**. These tests are executed on the [deployed version of the app](#), mimicking how a user would interact with the platform in a live environment. Each test scenario covers core features, helping to ensure a stable and user-friendly experience.

Planned tests include:

- Adding a movie with all required fields
- Editing and deleting movie entries
- Filtering movies by genre and status
- Marking a movie as watched and verifying UI update
- Adding a new category
- Editing and deleting movie categories



- Soring the whole watchlist

## Static Code Analysis

Static analysis will be conducted using **SonarQube** and **SonarScanner**, executed locally for backend code. Key metrics that will be collected are:

- Code smells and maintainability index
- Detection of unused code or poor practices
- Security rule violations and duplication

This analysis will tell whether application satisfies general clean code standards and passes all critical SonarQube rules without blocking issues.

## Code Coverage Analysis

Code coverage analysis will be conducted using **JaCoCo**, which is integrated within the IntelliJ IDE. The analysis primarily targets **backend unit and integration tests**, with a focus on critical components such as the MovieService, UserService, and REST controller classes. The resulting reports will be generated and reviewed in IntelliJ, providing visual insights into which methods and branches were executed during testing.

## Entry and Exit Criteria

The following table outlines the entry and exit criteria for each testing phase used in this project. This table is necessary because it tells us which specific conditions must be met to ensure that the testing environment is ready and that tests can be executed effectively. Similarly, each phase defines exit criteria that must be satisfied to consider the phase successfully completed.



Test Phase	Entry Criteria	Exit Criteria
Unit Testing	Backend builds without compilation errors	All service-level tests pass with $\geq$ 80% coverage
Integration	Database schema is initialized, endpoints are active	All major API flows return expected status/results
System Testing	Frontend deployed and Selenium scripts configured	All UI flows work without crashes or regressions
Static Analysis	Project is committed to GitHub	No major violations or security flaws remain

## Testing Environment and Tools

The Movies Watchlist application was tested using a combination of automated and manual techniques, supported by the following tools and technologies:

Component	Tool / Technology	Purpose
Development IDE	IntelliJ IDEA	Java backend development and test execution

Build Tool	Maven	Dependency management and project builds
Programming Languages	Java, JavaScript (React)	Backend and frontend development
Unit Testing	JUnit (Java Spring Boot)	Automated testing of backend logic
Integration Testing	JUnit with Spring MockMvc	API-level testing of controller-service flow
System Testing	Selenium WebDriver (Java)	Frontend UI testing through user simulation
Static Code Analysis	SonarQube (locally hosted)	Code quality checks, rule enforcement
Browser for Tests	Google Chrome	Running and inspecting Selenium test sessions
Version Control	GitHub	Codebase hosting, versioning, collaboration
Hosting Provider	Render	Live deployment of frontend and backend



## Deliverables

Throughout the SVVT project lifecycle, several testing artifacts are going to be produced to document and support the validation process. These deliverables are essential for demonstrating testing coverage, methodology, and outcomes.

The following items represent the primary deliverables of the verification and validation process:

- Unit test source files (JUnit)
- Selenium scripts
- Code coverage with JaCoCo
- Static code analysis summary from SonarQube
- Screenshots of test execution and passed cases
- Final summary report

## Test Execution

### Unit Testing

Unit testing in this project was conducted using the **JUnit 5** framework, with extensive use of **Mockito** for mocking dependencies. These tests validated the internal behavior of the backend logic and ensured that each method behaves correctly in isolation. The objective was to confirm that:

- Repositories correctly return mock data
- Services correctly process logic based on mocked inputs
- Controllers return proper HTTP responses based on mocked services

### Scope of Unit Tests

The test suite includes classes that cover all core backend components:

- **Repository Layer** – basic mocking coverage to validate DAO methods
- **Service Layer** – full coverage with all CRUD operations, business logic, and input validation



- **Controller Layer** – endpoint-level behavior and HTTP response validation using MockMvc

Unit test methods are completely isolated from the live database. All repository interactions are mocked to preserve data integrity and prevent side effects during testing.

### Code Coverage

As observed from the code coverage summary (see attached screenshot), backend unit testing achieved:

- **~85% method coverage**
- **Over 80% branch and line coverage** for key services
- Most tested classes: MovieService, UserService, WatchlistGroupService, and MovieController

Tests such as:

- testMarkAsWatched\_Success() and testMarkAsWatched\_AlreadyWatched()
- testAddMovie(), testEditMovie\_Simple()
- testFilterMoviesByStatus(), testSortMoviesByWatchlistOrderAsc()

Are examples that the small parts of the code (unit logic) work correctly in different situations, including unusual cases and when errors happen.

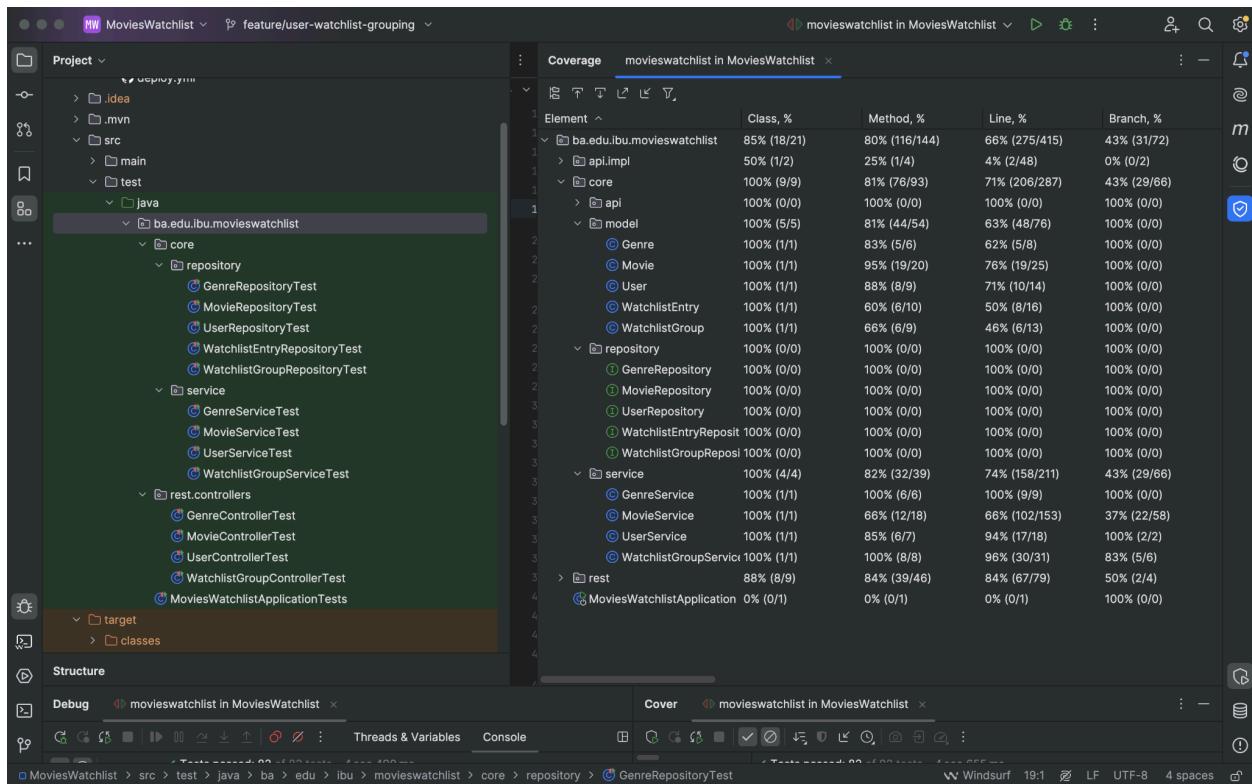
To conclude, all unit tests passed successfully with expected assertions. The codebase is highly stable and internally consistent based on the tested business rules.

### Artifacts produced in this part are:

- JUnit test source code (organized by layer)
- Screenshot of IntelliJ code coverage summary



- Test logs from console output



## Integration Testing

Integration testing was conducted to verify that various backend components—such as controllers, services, and repositories—communicate correctly and produce the expected results when working together. These tests are written using JUnit and Spring's testing support, including `@SpringBootTest` and `MockMvc`.

## Scope of Integration Tests

The tests focus on validating the behavior of complete request flows for:

- Creating and editing movies
- Marking movies as watched
- Filtering and sorting via endpoint parameters
- Managing watchlist categories
- Toggling notification settings



Rather than mocking all dependencies, integration tests rely on real service and controller layers while mocking the database (using `@DataJpaTest`), in order not to change actual data.

## Test Objectives

- Ensure that HTTP requests made to the REST API result in the expected logic being executed
- Confirm that service methods and validation are triggered properly when requests pass through controllers
  - Validate edge cases and negative scenarios (e.g., missing parameters, invalid input, or nonexistent IDs)

## Example Test Flows

- Sending a POST `/api/movies` request with valid payload and checking that the movie is persisted and a 201 status is returned
- Updating an existing movie using PUT `/api/movies/{id}` and asserting that changes are saved correctly
- Using GET `/api/movies/filter?status=Watched` to ensure the filtering logic works end-to-end

In the end, all integration tests passed successfully and confirmed that the system correctly handled expected and edge-case user actions. Test cases validated status codes, returned content structures, and method execution paths across all major backend workflows.

## Artifacts produced in this part are:

- Integration test source files (JUnit with Spring Boot)
- Console output showing successful end-to-end test execution

38 tests that I had in service layer, alongside 21 test in the controllers layer have successfully passed. The screenshots attached below serve as an evidence for that. Additionally, [here](#) in the GitHub repository for backend in folders **core/services** and **rest/controllers**, you can analyze the code for these tests in detail.

Project ▾

java

ba.edu.ibu.movieswatchlist

core

repository

GenreRepositoryTest

MovieRepositoryTest

UserRepositoryTest

GenreRepositoryTest.java

```
import ...
class GenreRepositoryTest { ...
    @Mock 10 usages
    private GenreRepository genreRepository;
}
```

Run controllers in MoviesWatchlist

controllers (ba.edu.ibu.movieswatchlist.rest)

- UserControllerTest
  - testAddUser()
  - testChangeNotificationStatus()
- GenreControllerTest
  - testAddGenre()
  - testGetAllGenres()
  - testSuggestGenre()
- MovieControllerTest
  - testDeleteMovie()
  - testEditMovie()
  - testCreateMovie()
  - testGetMoviesByUser()
  - testFilterMoviesByGenre()
  - testFilterMoviesByWatchlistOrder()
  - testFilterMoviesByStatus()
  - testSortMoviesByWatchlistOrder()
  - testMarkMovieAsWatched()
- WatchlistGroupControllerTest

Windsurf 19:1 LF UTF-8 4 spaces

Project ▾

java

ba.edu.ibu.movieswatchlist

core

GenreRepositoryTest.java

```
@Mock 10 usages
private GenreRepository genreRepository;

```

Run service in MoviesWatchlist

service (ba.edu.ibu.movieswatchlist.core)

- GenreServiceTest
  - testGetGenreByName\_GenreExists()
  - testAddGenre()
  - testGetAllGenres()
  - testGetGenreById\_GenreDoesNotExist()
  - testSuggestGenre()
  - testGetGenreById\_GenreExists()
  - testGetGenreByName\_GenreDoesNotExist()
- MovieServiceTest
  - testSortMoviesByWatchlistOrderInvalid()
  - testGetMovieById()
  - testDeleteMovie()
  - testGetMoviesByUser()
  - testAddMovie()
  - testGetMoviesByUserSortedByTitle()
  - testFilterMoviesByGenre()
    - testMarkAsWatched\_AlreadyWatched()
    - testEditMovie\_Simple()
    - testFilterMoviesByStatus()
    - testGetMovieById\_NotFound()
    - testMarkAsWatched\_Success()

Windsurf 19:1 LF UTF-8 4 spaces

## System Testing (Selenium)

System testing was implemented using **Selenium WebDriver** to automate real user interactions with the deployed application. Code for the described tests below is available in this [GitHub Repository here.](#)

### Test Case 1 - Add a movie with valid data

<b>Test Name:</b> Add Movie Modal – Positive Flow Test				
<b>Description:</b> This test verifies that a user can successfully open the Add Movie Modal, enter valid data into the movie form fields, and submit the form to add a new movie to the list.				
<b>Pre-condition(s):</b> - User is logged in - Homepage is fully loaded				
<b>Test Steps</b>	<b>Test Data</b>	<b>Expected Result</b>	<b>Actual Result</b>	<b>Status</b>
1. Click on the “Add Movie Modal” button 2. Fill in all required fields: – Title – Description – Category (new + dropdown) – Watchlist Order – Genre 3. Click on “Add Movie” button 4. Verify that the movie is added to the list and appears with correct data	Title: Interstellar Description: A sci-fi movie about space travel Category: Sci-Fi Watchlist Order: Next Up Genre: Adventure Dropdown Category: Waiting Movie Premiere	The movie Interstellar is successfully added and displayed in the user's watchlist with all the entered details	The movie Interstellar appears in the list with correct title, description, genre, and watchlist information	PASS



```

EXPLORER      ...
OPEN EDITORS   ...
SVT...  ...
core
config
driver-setup.ts
data
data.json
page-objects
base-page.ts
landing-page.ts
login-page.ts
sneakers-page.ts
node_modules
tests
jest.config.js
package-lock.json
package.json
README.md

TS driver-setup.ts  {} data.json  TS 1-addMovie.test.ts X  TS landing-page.ts  TS login-page.ts

tests > TS 1-addMovie.test.ts > ...
1 import { HomePage } from "../core/page-objects/landing-page";
2 import { LoginPage } from "../core/page-objects/login-page";
3 import { Builder, By, WebDriver } from "selenium-webdriver";
4 import { createDriver, quitDriver } from "../core/config/driver-setup";
5 import { readFileSync } from "fs";
6 import * as path from "path";
7
8 const dataFilePath = path.resolve(__dirname, "../core/data/data.json");
9 const testData = JSON.parse(readFileSync(dataFilePath, "utf8"));
10
11 let driver: WebDriver;
12 let HomePage: HomePage;
13 let LoginPage: LoginPage;
14
15 beforeEach(async () => {
16   driver = await createDriver(testData.url.home_page);
17   HomePage = new HomePage(driver);
18   LoginPage = new LoginPage(driver);
19
20 }, 3000);
21
22 test("add movie", async () => {
23   await LoginPage.provideEmail();
24   await LoginPage.clickOnLoginButton();
25   await HomePage.openAddMovieModal();
26   await HomePage.fillMovieForm();
27 }, 50000);
28

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

> svt-selenium@1.0.0 test
> jest --runInBand 1-addMovie.test.ts

ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModuleInterop` to file (usually `tsconfig.json`). See https://blogs.msdn.microsoft.com/typescript/2018/01/31/announcing-typescript-2-7/#easier-ecm
information.
PASS  tests/1-addMovie.test.ts (16.472 s)
  ✓ add movie (13629 ms)

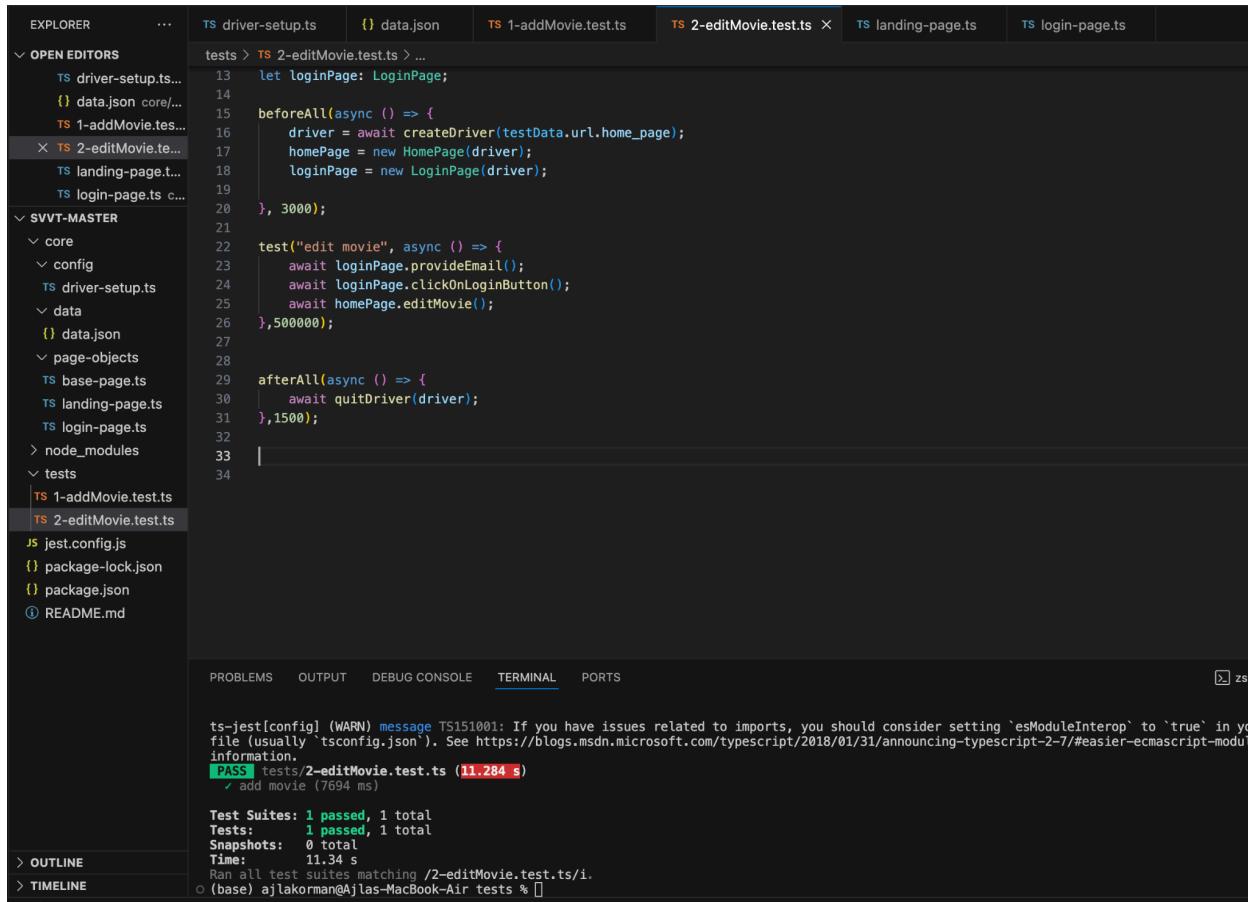
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        16.523 s, estimated 18 s
Ran all test suites matching /1-addMovie.test.ts/i.
(base) ajlakorman@Ajlas-MacBook-Air tests % npm run test 1-addMovie.test.ts

```

## Test Case 2 - Edit an existing movie

<b>Test Name:</b> Edit Movie – Update Description and Genre				
<b>Description:</b> This test verifies that a user can successfully edit an existing movie's description and genre using the UI and save the updated data.				
<b>Pre-condition(s):</b> - A movie must exist in the list - User is logged in and on the homepage				
Test Steps	Test Data	Expected Result	Actual Result	Status
1. Click the edit icon on a movie card 2. Clear and enter a new	New Description: Rewatch during break – now with friends and	The movie is updated with the new description and genre, and changes are	Description and genre were successfully updated and are visible in the	PASS

description 3. Open genre dropdown and select “Comedy” 4. Click “Save Changes” 5. Verify updated values appear on the movie card	snacks. New Genre: Comedy	reflected in the UI	watchlist	
---	---------------------------------	---------------------	-----------	--



The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure with files like `driver-setup.ts`, `data.json`, `1-addMovie.test.ts`, `2-editMovie.test.ts`, `landing-page.ts`, and `login-page.ts`.
- CODE EDITOR:** The file `2-editMovie.test.ts` is open, displaying Jest test code for editing a movie.
- TERMINAL:** Shows the output of the Jest test run, indicating 1 passed test in 11.284 s.
- STATUS BAR:** Shows the status `(base) ajlakorman@Ajlas-MacBook-Air tests %`.

```

 13   let LoginPage: LoginPage;
 14
 15   beforeEach(async () => {
 16     driver = await createDriver(testData.url.home_page);
 17     homePage = new HomePage(driver);
 18     LoginPage = new LoginPage(driver);
 19
 20   }, 3000);
 21
 22   test("edit movie", async () => {
 23     await LoginPage.provideEmail();
 24     await LoginPage.clickOnLoginButton();
 25     await homePage.editMovie();
 26   }, 50000);
 27
 28   afterEach(async () => {
 29     await quitDriver(driver);
 30   }, 1500);
 31
 32
 33
 34

```

```

ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModuleInterop` to `true` in your file (usually `tsconfig.json`). See https://blogs.msdn.microsoft.com/typescript/2018/01/31/announcing-typescript-2-7/#easier-ecmascript-module-information.
PASS  tests/2-editMovie.test.ts (11.284 s)
  ✓ add movie (7694 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        11.34 s
Ran all test suites matching /2-editMovie.test.ts/i.
(base) ajlakorman@Ajlas-MacBook-Air tests %

```

**Test Case 3 - Delete a Movie**

<b>Test Name:</b> Delete Movie – Confirmation Modal Flow				
<b>Description:</b> This test ensures that a movie can be deleted using the trash icon, and that the confirmation modal functions correctly.				
<b>Pre-condition(s):</b> - A movie exists in the watchlist - User is logged in and on the homepage				
Test Steps	Test Data	Expected Result	Actual Result	Status
1. Click the delete icon on the movie card 2. Wait for the confirmation modal 3. Click the “Delete” button in the modal 4. Confirm that the movie is removed from the list	Any existing movie in the user's watchlist	The movie is removed from the interface and no longer appears in the list	The movie is successfully deleted after confirmation	PASS



The screenshot shows a terminal window within VS Code displaying Jest test results. The command run was `npx jest tests/3-deleteMovie.test.ts`. The output shows 1 passed test named "edit movie" which took 7.658 seconds. The terminal also displays TypeScript configuration information and a warning about esModuleInterop.

```

ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModuleInterop` file (usually `tsconfig.json`). See https://blogs.msdn.microsoft.com/typescript/2018/01/31/announcing-typescript-2-7/#esModuleInterop.
PASS  tests/3-deleteMovie.test.ts (7.658 s)
  ✓ edit movie (3777 ms)

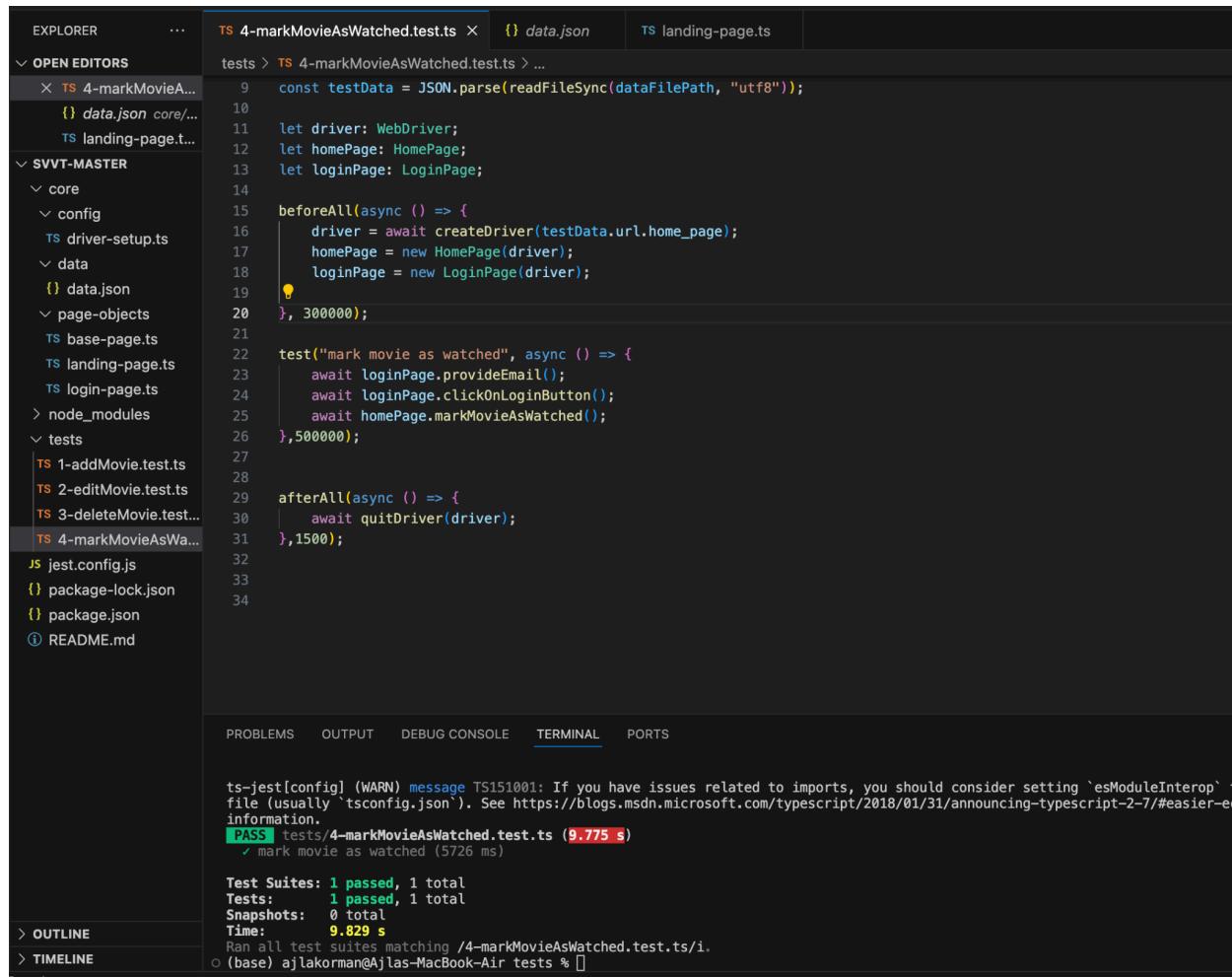
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        7.711 s, estimated 8 s
Ran all test suites matching /3-deleteMovie.test.ts/i.

```

## Test Case 4 - Mark Movie as Watched

<b>Test Name:</b> Mark Movie as Watched – Status Change Verification				
<b>Description:</b> This test confirms that a movie marked as “To Watch” can be updated to “Watched” through the UI, and that the change is reflected correctly.				
<b>Pre-condition(s):</b> - Movie with status “To Watch” exists - User is logged in and on homepage				
Test Steps	Test Data	Expected Result	Actual Result	Status
1. Read current movie status from UI 2. Click on the	Any movie card where status = “To Watch”	Status value updates from “To Watch” to “Watched” after	Movie status changed successfully and reflected on UI	PASS

check icon (“mark as watched”) 3. Confirm action in modal 4. Wait for status to update 5. Verify the status now reads “Watched”		confirmation		
--	--	--------------	--	--



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar showing project files like `4-markMovieAsWatched.test.ts`, `data.json`, and `landing-page.ts`. The main area displays the contents of `4-markMovieAsWatched.test.ts`:

```

TS 4-markMovieAsWatched.test.ts × {} data.json TS landing-page.ts
tests > TS 4-markMovieAsWatched.test.ts > ...
9  const testData = JSON.parse(readFileSync(dataFilePath, "utf8"));
10
11 let driver: WebDriver;
12 let homePage: HomePage;
13 let loginPage: LoginPage;
14
15 beforeAll(async () => {
16   driver = await createDriver(testData.url.home_page);
17   homePage = new HomePage(driver);
18   loginPage = new LoginPage(driver);
19   await loginPage.waitForElement();
20 }, 300000);
21
22 test("mark movie as watched", async () => {
23   await loginPage.provideEmail();
24   await loginPage.clickOnLoginButton();
25   await homePage.markMovieAsWatched();
26 }, 500000);
27
28
29 afterAll(async () => {
30   await quitDriver(driver);
31 }, 1500);
32
33
34

```

At the bottom, a terminal window shows the test results:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModuleInterop` to true in your tsconfig.json. See https://blogs.msdn.microsoft.com/typescript/2018/01/31/announcing-typescript-2-7/#easier-import-resolution.
PASS  tests/4-markMovieAsWatched.test.ts (9.775 s)
  ✓ mark movie as watched (5726 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        9.829 s
Ran all test suites matching /4-markMovieAsWatched.test.ts/i.
(base) ajlakorman@Ajas-MacBook-Air tests %

```

Icons for `OUTLINE` and `TIMELINE` are visible at the bottom left.

**Test Case 5 - Apply Filters**

<b>Test Name:</b> Filter Movies – Genre, Status, and Category				
<b>Description:</b> This test ensures that the user can apply filters based on genre, status, and watchlist category using the filter dropdowns and see a filtered list.				
<b>Pre-condition(s):</b> - User is logged in - Movies exist in the list covering various genres, statuses, and categories				
Test Steps	Test Data	Expected Result	Actual Result	Status
1. Open genre dropdown and select “Action” 2. Open status dropdown and select “Watched” 3. Open watchlist category dropdown and select “Waiting Movie Premiere” 4. Click “Apply Filters” 5. Confirm that the filtered results reflect the selected criteria	Genre: Action Status: Watched Category: Waiting Movie Premiere	The movie list updates to only show items that match all selected filters	Movie list was successfully filtered based on genre, status, and category	PASS



```

const testData = JSON.parse(readFileSync(dataFilePath, "utf8"));
let driver: WebDriver;
let HomePage: HomePage;
let LoginPage: LoginPage;

beforeAll(async () => {
  driver = await createDriver(testData.url.home_page);
  HomePage = new HomePage(driver);
  LoginPage = new LoginPage(driver);
}, 300000);

test("filter movies", async () => {
  await LoginPage.provideEmail();
  await LoginPage.clickOnLoginButton();
  await HomePage.applyFilters();
}, 50000);

afterAll(async () => {
  await quitDriver(driver);
}, 1500);

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModuleInterop` file (usually `tsconfig.json`). See <https://blogs.microsoft.com/typescript/2018/01/31/announcing-typescript-2-7/#easier-information>.

PASS tests/5-filterMovies.test.ts (13.63 s)

✓ filter movies (10144 ms)

Test Suites: 1 passed, 1 total  
 Tests: 1 passed, 1 total  
 Snapshots: 0 total  
 Time: 13.678 s  
 Ran all test suites matching /5-filterMovies.test.ts/i.  
 (base) ajlakorman@Ajlas-MacBook-Air tests %

Ln 22, Col 20 Spaces: 4

## Test Case 6 - Apply Sorting

Test Name: Sort Movies – Watchlist Order Ascending				
Description: This test checks if movies can be sorted by watchlist order in ascending sequence using the sorting dropdown and Apply Filters button.				
Pre-condition(s):				
<ul style="list-style-type: none"> <li>- User is logged in</li> <li>- Watchlist contains multiple movies with varying watchlist positions</li> </ul>				
Test Steps	Test Data	Expected Result	Actual Result	Status
1. Open the sort dropdown 2. Select “Watchlist Order”	Sort option: Watchlist Order (Asc)	Movies are reordered in ascending watchlist order	Movies appeared in the expected order, starting from "Next Up"	PASS



(Asc)" 3. Click "Apply Filters" 4. Verify the movie list is reordered accordingly		on the UI	or lowest rank	
---	--	-----------	----------------	--

```

EXPLORER      ...      TS 4-markMovieAsWatched.test.ts      TS landing-page.ts      TS 6-sortMovies.test.ts
OPEN EDITORS
  TS 4-markMovieAs...
  TS landing-page.t...
  × TS 6-sortMovies.t...
SVVT-MASTER
  core
    > config
    > data
    > page-objects
      TS base-page.ts
      TS landing-page.ts
      TS login-page.ts
    > node_modules
    > tests
      TS 1-addMovie.test.ts
      TS 2-editMovie.test.ts
      TS 3-deleteMovie.test...
      TS 4-markMovieAsWa...
      TS 5-filterMovies.test.ts
      TS 6-sortMovies.test.ts
    JS jest.config.js
    {} package-lock.json
    {} package.json
    README.md

tests > TS 6-sortMovies.test.ts > test("filter movies") callback
  9  const testData = JSON.parse(readFileSync(dataFilePath, "utf8"));
 10
 11 let driver: WebDriver;
 12 let HomePage: HomePage;
 13 let LoginPage: LoginPage;
 14
 15 beforeEach(async () => {
 16   driver = await createDriver(testData.url.home_page);
 17   HomePage = new HomePage(driver);
 18   LoginPage = new LoginPage(driver);
 19
 20 }, 30000);
 21
 22 test("filter movies", async () => {
 23   await LoginPage.provideEmail();
 24   await LoginPage.clickOnLoginButton();
 25   await HomePage.applySorting();
 26 }, 500000);
 27
 28
 29 afterEach(async () => {
 30   await quitDriver(driver);
 31 }, 1500);
 32
 33
 34

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```

ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModuleInterop` (usually `tsconfig.json`). See https://blogs.msdn.microsoft.com/typescript/2018/01/31/announcing-typescript-2-7/#es-module-imports
PASS  tests/6-sortMovies.test.ts (8.718 s)
  ✓ Filter movies (5379 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        8.761 s
Ran all test suites matching /6-sortMovies.test.ts/i.
(base) ajlakorman@jlas-MacBook-Air tests % []

```

Ln 25, Col 32    Space

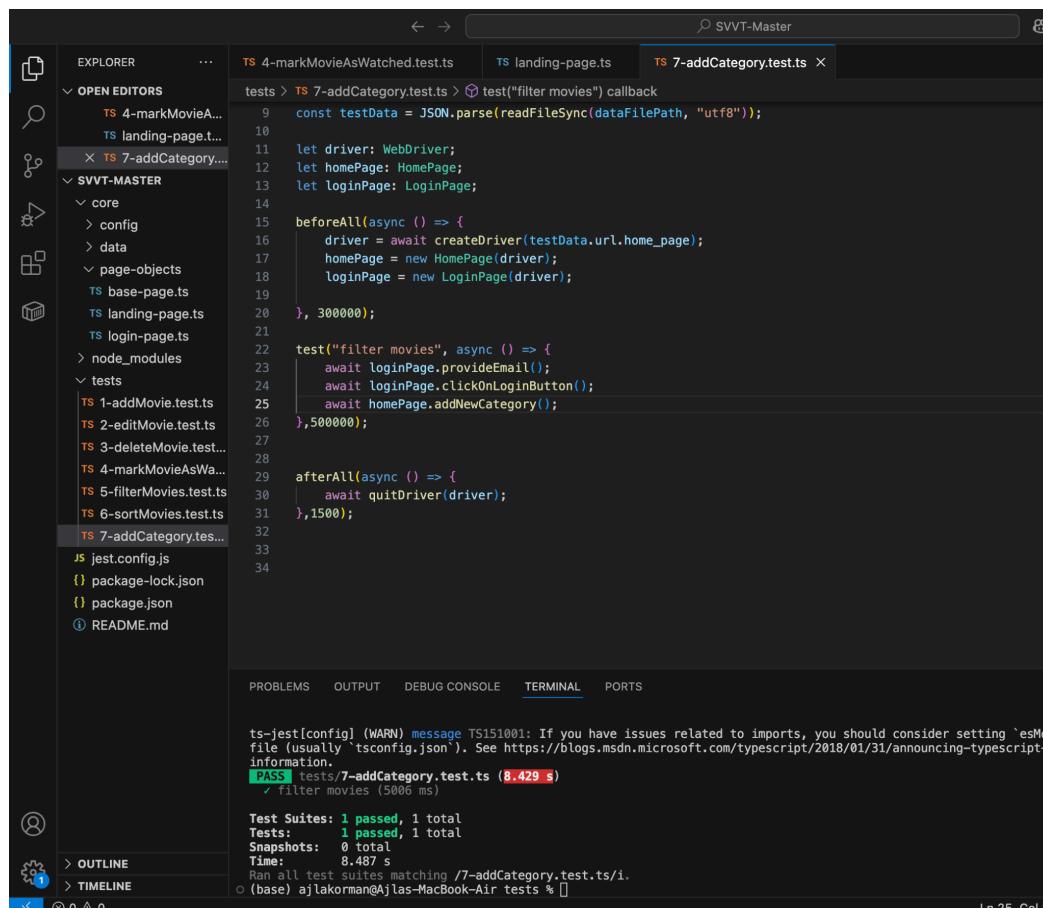
## Test case 7 - Add New Category

<b>Test Name:</b> Add New Watchlist Category
<b>Description:</b> This test verifies that a user can create a new watchlist category using the "Add New Category" modal.



**Pre-condition(s):** - User is logged in  
 - Category with the same name does not already exist

Test Steps	Test Data	Expected Result	Actual Result	Status
1. Click on “Add New Category” button 2. Enter a new category name 3. Click the “Add” button 4. Wait for UI to update and confirm category appears in dropdown	New Category: "Holiday Picks"	The new category “Holiday Picks” is added and becomes available for selection in the category dropdown	New category successfully appeared and was selectable from the list	PASS



```

EXPLORER      TS 4-markMovieAsWatched.test.ts  TS landing-page.ts  TS 7-addCategory.test.ts ×
tests > TS 7-addCategory.test.ts > ⚡ test("filter movies") callback
  9  const testData = JSON.parse(readFileSync(dataFilePath, "utf8"));
 10 
 11 let driver: WebDriver;
 12 let homePage: HomePage;
 13 let loginPage: LoginPage;
 14 
 15 beforeEach(async () => {
 16   driver = await createDriver(testData.url.home_page);
 17   homePage = new HomePage(driver);
 18   loginPage = new LoginPage(driver);
 19 }, 300000);
 20 
 21 test("filter movies", async () => {
 22   await loginPage.provideEmail();
 23   await loginPage.clickOnLoginButton();
 24   await homePage.addNewCategory();
 25 }, 50000);
 26 
 27 afterEach(async () => {
 28   await quitDriver(driver);
 29 }, 1500);
 30 
 31 }, 1500);
 32 
 33 }
 34

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModule` file (usually `tsconfig.json`). See https://blogs.msdn.microsoft.com/typescript/2018/01/31/announcing-typescript-2.9-information.
PASS  tests/7-addCategory.test.ts (8.429 s)
  ✓ filter movies (5006 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        8.487 s
Ran all test suites matching /7-addCategory.test.ts/i.
(base) ajlakorman@Ajas-MacBook-Air tests % []

```

## Test Case 8 - Edit Category

<b>Test Name:</b> Edit Watchlist Category – Rename Existing Category				
<b>Description:</b> This test validates that an existing watchlist category name can be updated through the edit category modal.				
<b>Pre-condition(s):</b> - Category “Watch during Winter Holidays” exists - User is logged in				
Test Steps	Test Data	Expected Result	Actual Result	Status
1. Click on “Edit Category” button 2. Open the dropdown and select “ <b>Watch during Winter Holidays</b> ” 3. Enter a new name: “Holiday Picks” 4. Click on “Save Changes” 5. Verify the name is updated and shown in dropdown	Old Name: Watch during Winter Holidays  New Name: Holiday Picks	Category name is successfully updated and reflected in the UI	The selected category was renamed to “Holiday Picks” and appeared updated in the dropdown	PASS



```

EXPLORER      ...   TS 4-markMovieAsWatched.test.ts    TS 8-editCategory.test.ts ×   TS landing-page.ts   TS 7-addCategory.test.ts
OPEN EDITORS
  ✓ OPEN EDITORS
    TS 4-markMovieA...
    ✘ TS 8-editCategory... (active)
    TS landing-page.t...
    TS 7-addCategory...
  ✓ SVVT-MASTER
    ✓ core
    > config
    > data
    ✓ page-objects
      TS base-page.ts
      TS landing-page.ts
      TS login-page.ts
    > node_modules
    ✓ tests
      TS 1-addMovie.test.ts
      TS 2-editMovie.test.ts
      TS 3-deleteMovie.test.ts
      TS 4-markMovieAsWa...
      TS 5-filterMovies.test.ts
      TS 6-sortMovies.test.ts
      TS 7-addCategory.tes...
      TS 8-editCategory.tes...
    JS jest.config.js
    {} package-lock.json
    {} package.json
    README.md

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
Snapshots: 0 total
Time: 6.642 s, estimated 21 s
Ran all test suites matching /8-editCategory.test.ts/i.
● (base) ajlakorman@Ajlas-MacBook-Air tests % clear
● (base) ajlakorman@Ajlas-MacBook-Air tests % npm run test 8-editCategory.test.ts
> svvt-selenium@1.0.0 test
> jest --runInBand 8-editCategory.test.ts
ts-jest[config] (WARN) message TS151001: If you have issues related to imports, you should consider setting `esModule` file (usually `tsconfig.json`). See https://blogs.msdn.microsoft.com/typescript/2018/01/31/announcing-typescript-2-7-information.
PASS  tests/8-editCategory.test.ts (10.352 s)
  ✓ edit category (7139 ms)

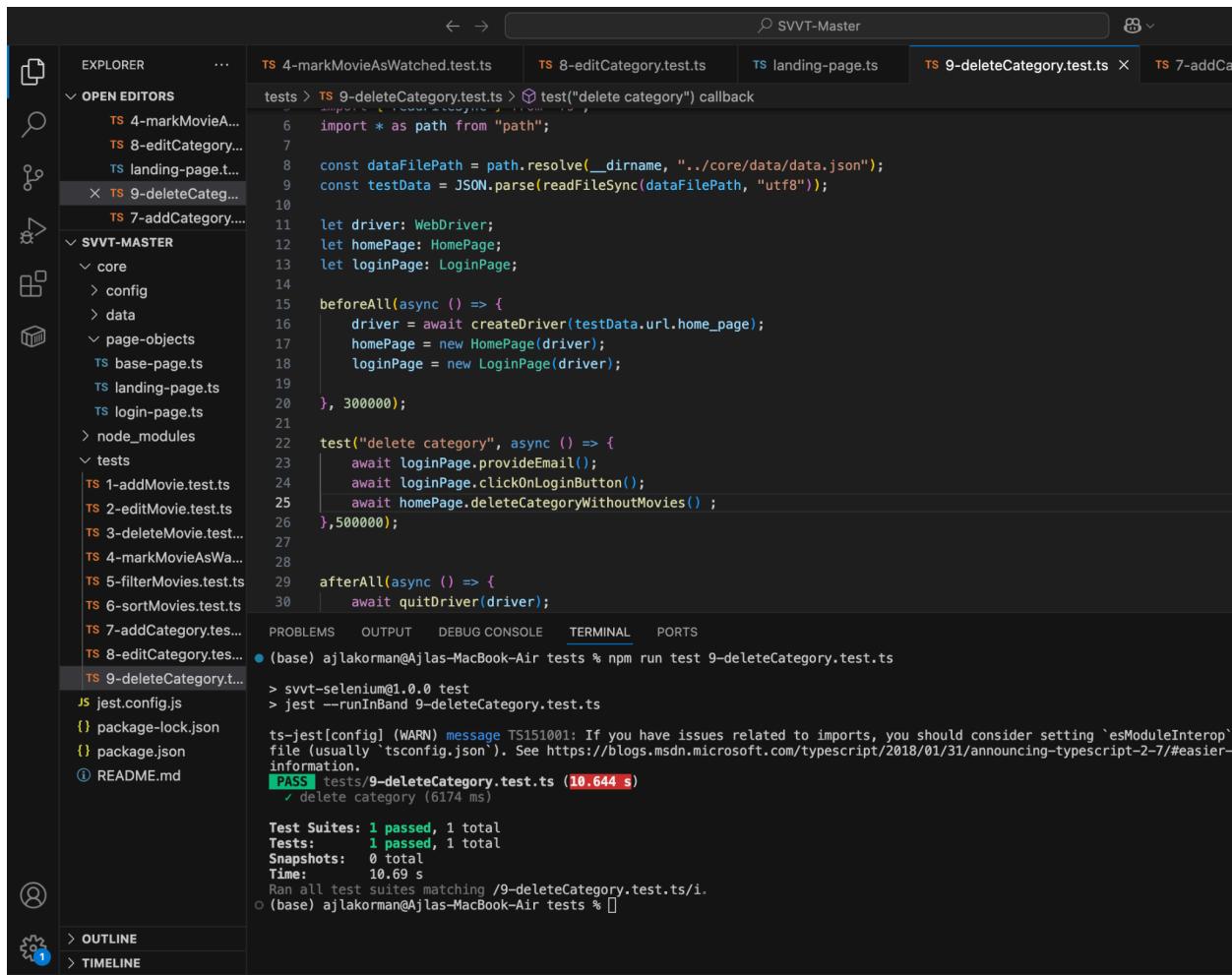
Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 10.411 s
Ran all test suites matching /8-editCategory.test.ts/i.
○ (base) ajlakorman@Ajlas-MacBook-Air tests % []

```

## Test Case 9 - Delete Category

<b>Test Name:</b> Delete Watchlist Category – Without Associated Movies				
<b>Description:</b> This test ensures that a watchlist category can be deleted even if it contains movies, by choosing to delete the category without its associated movies.				
<b>Pre-condition(s):</b> - Category “Holiday Best Picks” exists - User is logged in				
Test Steps	Test Data	Expected Result	Actual Result	Status
1. Click on “Delete Category” button	Category to delete: Holiday Best Picks	Category is deleted while keeping the movies in the	Category removed successfully and UI reflects	PASS

2. Select “Holiday Best Picks” from dropdown 3. Click on “Delete Without Movies” button 4. Verify that the category no longer appears in the category list		watchlist under no category	change	
--	--	-----------------------------	--------	--



The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure with files like 4-markMovieAsWatched.test.ts, 8-editCategory.test.ts, landing-page.ts, 9-deleteCategory.test.ts (which is the active file), and 7-addCategory.test.ts.
- Code Editor:** Displays the code for 9-deleteCategory.test.ts. The code uses Jest and TypeScript to test the deletion of a category. It imports path, creates a WebDriver instance, and performs a test to delete a category without movies. The test passes with a duration of 10.644 seconds.
- Terminal:** Shows the command `npm run test 9-deleteCategory.test.ts` being run, and the output of the test execution.

```

tests > TS 9-deleteCategory.test.ts > test("delete category") callback
  1 import * as path from "path";
  2
  3 const dataFilePath = path.resolve(__dirname, "../core/data/data.json");
  4 const testData = JSON.parse(readFileSync(dataFilePath, "utf8"));
  5
  6 let driver: WebDriver;
  7 let homePage: HomePage;
  8 let loginPage: LoginPage;
  9
 10 beforeEach(async () => {
 11   driver = await createDriver(testData.url.home_page);
 12   homePage = new HomePage(driver);
 13   loginPage = new LoginPage(driver);
 14
 15 }, 300000);
 16
 17 test("delete category", async () => {
 18   await loginPage.provideEmail();
 19   await loginPage.clickOnLoginButton();
 20   await homePage.deleteCategoryWithoutMovies();
 21 }, 500000);
 22
 23 afterEach(async () => {
 24   await quitDriver(driver);
 25 });
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1087
 1088
 1089
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1097
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1138
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1158
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1168
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1187
 1188
 1189
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1197
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1228
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1238
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1258
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1268
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1277
 1278
 1279
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1287
 1288
 1289
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1297
 1298
 1299
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1328
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1368
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1378
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1387
 1388
 1389
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1408
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1428
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1477
 1478
 1479
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1487
 1488
 1489
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1498
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1508
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1518
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1528
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1558
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1568
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1578
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1587
 1588
 1589
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1598
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1608
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1618
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1628
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1638
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1658
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1668
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1678
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1687
 1688
 1689
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1698
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1708
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1718
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1728
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1738
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1759
 1760
 1761
 1762
 1763
 17
```

## Static Code Analysis

SonarQube was used for static code analysis in this project. It is a widely used static code analysis tool that automatically reviews code without executing it. It is used in professional software development teams to enforce clean coding standards and ensure long-term code quality and consistency. Its primary purpose is to detect:

- Code smells (inefficient coding practices)
- Security vulnerabilities
- Reliability issues (bugs that could cause runtime failures)
- Code duplications
- Overall maintainability issues

SonarScanner is the command-line tool that analyzes your code and sends the results to a SonarQube server for processing and visualization. It reads configuration from the sonar-project.properties file placed in the project root directory, and it includes:

- Project name and version
- Source and test code locations
- Compilation output path (target/classes)

In this project, SonarQube Community Edition was installed and run locally, and was used to analyze the Java Spring Boot backend of the Movie Watchlist application.

The process was as follows:

1. SonarQube server was started locally and accessed via <http://localhost:9000>
2. A configuration file named sonar-project.properties was created in the project root, specifying:
  - The source directory: src/main/java
  - The test directory: src/test/java
  - Project metadata
3. The project was compiled using Maven (mvn clean install)
4. SonarScanner was then executed to start the analysis



5. The results were displayed in the SonarQube dashboard.

### Analysis Results

The analysis was successful, and the project was automatically created in the SonarQube UI under the name Movie Watchlist Backend. The following metrics were collected and are summarized in the screenshots below.

Category	Result	Explanation
Security	A (0 issues)	No vulnerabilities or security hotspots detected
Reliability	A (0 bugs)	No logic errors or runtime failures identified
Maintainability	A (17 code smells)	Minor issues that do not affect functionality but could reduce code readability and maintainability
Duplications	0.0%	No duplicated blocks of code detected
Coverage	0.0%	Test coverage data was not imported during this run. IntelliJ's internal coverage via JaCoCo is documented separately



SonarQube community Projects Issues Rules Quality Profiles Quality Gates Administration More ▾

⚠️ Embedded database should be used for evaluation purposes only. It doesn't support scaling, upgrading to a new SonarQube Server version, or migration to another database engine. [Learn more](#)

☆ Movie Watchlist Backend / main ✓ ?

Overview Issues Security Hotspots Code Measures Activity Project Settings ▾ Project Information

To benefit from more of SonarQube Community Build's features, [set up analysis in your favorite CI](#).

**main** 1.2k Lines of Code • Version 1.0 • Set as homepage Last analysis 2 minutes ago

**Passed** The last analysis has warnings. [See details](#)

New Code Overall Code

Security	Reliability	Maintainability
0 Open issues	0 Open issues	17 Open issues

The last analysis has warnings. [See details](#)

New Code Overall Code

Security	Reliability	Maintainability
0 Open issues	0 Open issues	17 Open issues

Accepted issues 0 Valid issues that were not fixed

Coverage 0.0% On 458 lines to cover.

Duplications 0.0% On 1.6k lines.

Security Hotspots 0

## Code Coverage Analysis

Code coverage was obtained using IntelliJ IDEA's built-in integration with JaCoCo, which displays results inline and supports exporting reports if needed. Coverage was measured for unit and integration tests across key backend layers, including:

- MovieService.java (business logic)
- UserService.java (authentication and profile updates)



- REST Controllers (such as MovieController, UserController)

IntelliJ **generated the code coverage report** inside of the IDE using a summary view. It provided enough information to confirm that a sufficient amount of code has been tested. The **focus was** on core business logic and user-facing API endpoints.

The final report indicated **85% of method coverage and 80% line and branch coverage**, satisfying the requirement for verifying tested behavior. A screenshot of the coverage summary was already included in the Unit Testing part.

## Conclusion

This project demonstrated the practical application of software verification, validation, and testing techniques on a real-world web application, Movie Watchlist, which consists of a Java Spring Boot backend and a React frontend. Throughout this project, industry-standard tools such as **JUnit**, **Selenium**, **SonarQube**, and **SonarScanner** were used to ensure software quality across multiple layers of the application.

## Entry and Exit Criteria Overview

Each testing phase was carefully aligned with defined **entry** and **exit** criteria:

- **Unit Testing** began once the backend compiled without errors. It concluded successfully with over 80% method-level coverage in service and controller classes.
- **Integration Testing** was initiated after the database schema was stabilized and endpoints were reachable. Exit was defined by verified API flows returning valid HTTP responses.
- **System Testing** using Selenium was executed after the frontend was deployed and stable. It concluded when critical UI scenarios (add/edit/delete/filter/sort/watch movies) passed consistently.



- **Static Analysis** started once the codebase was committed and IntelliJ builds succeeded. It exited when SonarQube returned a "Passed" quality gate with no critical issues.

## Deliverables Summary

The following deliverables were produced as part of this SVVT project:

- **Unit Test Code** (JUnit) covering repositories, services, and REST controllers
- **Selenium Test Scripts** automating real usage scenarios on the frontend
- **SonarQube Static Analysis Reports**, with screenshots included in the documentation
- **JaCoCo Coverage Summary** (within IntelliJ), showing high coverage rates
- **Test Plan Document**, describing the testing strategy, types, tools, environment, and coverage

## Project Summary

All phases of the software verification and validation cycle were successfully completed. No critical security, reliability, or logical issues were discovered during the process. Minor maintainability findings were noted by SonarQube but do not impact functionality. Additionally, automated frontend tests and backend assertions confirm that the core features of the system perform as expected.

By utilizing **open-source tools** and adhering to **defined testing workflows**, this project showcased that it can meet professional software quality standards. The result is a reliable, maintainable, and verifiable movie tracking system — ready for users and scalable for future development.