SE 3XA3: Test Report
Movie Guide

Team 35, PGH Software Solutions
Hamid Ghasemi and ghasemih
Pratyush Bhandari, bhandarp
Gazenfar Syed, syedg1

December 5, 2018

Contents

List of Tables

Table 1: **Revision History**

Date	**Version**	**Notes**
Dec3	1.0	Completed parts 1, 2, and 3
Dec4	1.1	Completed parts 4 and 7

1 Functional Requirements Evaluation

The aim of these tests is to make sure that the user is able to use the software according to the given requirements. These tests will include sorting, searching, and movie information access testing.

Test Name: FRSR
Results: The user is able to sort movies based on rating, popularity and release date.

Test Name: FRSE
Results: The user is able to search a movie by using movie name.

Test Name: FRAC
Results: The user is able to access movie's summary, rating and release date.

Test Name: FRTR
Results: The user is able to watch a movie trailer by simply selecting one of the list movie trailers for the movie.

2 Nonfunctional Requirements Evaluation

2.1 Usability

2.1.1 GUI Testing

The graphical user interface was tested by seven students from McMaster University who were interested in using our application and providing us with daily feedback on any issues they had encountered while using MovieGuide. Along with this, participants also observed the time it took them to perform requested tasks. At the end of the week, each participant gave a rating from 0 - 10 for the usability of the system. An average score of 7 was achieved at the end of the test.

Test Name: SS-1
Results: All participants were able to successfully complete installing the program on their phone within 2-3 minutes.

Test Name: SS-2
Results: All participants were able to successfully complete the task of searching a favourite movie by inputing the movie's name .

Test Name: SS-3
Results: All participants were able to successfully complete the task of sorting movies based on rating, release date and popularity.

Test Name: SS-4
Results: All participants mentioned that the tasks completed above were simple and intuitive. Their feedback was that MovieGuide was easy to install, easy to use, understandable and overall had a very intuitive user experience. All though all beta testers were satisfied, there were some suggestions for improvement. For instance, one of participants suggested that the application can have an option to sort the movies which will come out within 6 months. This way people can realize which movies are coming out.

2.1.2 Media Output Testing

The program was installed onto an Android phone. The results of attempting to launch the program were noted.

Test Name: SS-5
Results: The program installed on phones that has the Android OS and there were not any problem.

2.2 Performance

2.2.1 Screen Speed Performance

The performance was calculated based on how long the application takes to seamlessly perform user requests.

Test Name: SS-6
Results: The time to perform user requests was less than 2 second from all participants.

2.3 Output

2.3.1 Media Output Testing

The time between finding a movie and retrieving the data from the dataset was calculated. In addition to that, the aim of this test was to check if the output was the expected output. Each time a different movie is requested by a participant, the app must ensure that the movie info and the output file are consistent. In addition, the timeliness of receiving an API response of sorted movies was noted in this section.

Test Name: SS-7
Results: Movies were searched by participants are retrieved within approximately 3 seconds.

Test Name: SS-8
Results: Sorting movies based on rating, popularity and release date took about 5 seconds.

3 Comparison to Existing Implementation

In our app, we used retrofit which is an Android Studio Library. It is a powerful framework for authenticating and interacting with API's and sending network requests. Retrofit made our app easier to code since it can be used to create JSON objects and then we used that in our implementation, using it in our java implementation. This is one of the main difference between our code and the existing implementation.

4 Unit Testing

4.1 Load Movies API Unit Test

4.1.1 Control

Retrieve data from the API

4.1.2 Input

Query used to retrieve data from the database

4.1.3 Output

A list containing Movie objects. Each Movie object will contain parsed data from the API.

4.1.4 Procedure

A get request was made to the API using a query passed in as input. The data is parsed into Movie objects and a list of Movie objects is created. Finally, the list of Movie objects was iterated through and movie names were printed out to verify that data for that movie has been passed.

4.2 Sorting Movies Unit Test

4.2.1 Control

Returns a list of movies sorted based on input by making a query to the API.

4.2.2 Input

Select to sort by popularity, rating, or release date.

4.2.3 Output

The list of movies will be returned in sorted order based on input.

4.2.4 Procedure

A sorting option will be selected. The software will display the movies in sorted order. The information of the movies will be manually checked to determine whether the sorting has been done correctly.

4.3 Movie Trailers Unit Test

4.3.1 Control

Utilizes HTML5 to load the trailers into the program.

4.3.2 Input

The play button for the video is clicked.

4.3.3 Output

The program plays the selected video.

4.3.4 Procedure

A movie was selected from the displayed list of movies. One of the movie trailers was played. The response of the program was checked to determine whether the movie trailers functionality is functioning as expected.

5 Changes Due to Testing

5.1 Media Output Testing

Upon doing test SS-8 for media output testing, it was realized that our application was only sorting movies on the first page of results. Any movies after that were not sorted. This issue was later addressed by ensuring API calls reflected the specified sorting parameters for each page of our application.

5.2 GUI Testing

After conducting our tests for the GUI, it was concluded that no changes were needed.

5.3 Screen Speed Performance Testing

After running our tests for performance, it was conclude that no changed were needed.

6 Automated Testing

A major benefit of automated testing is that it allows one to perform a large number of tests in a very small time and in a consistent manner. Although, automated testing is a reliable way to test a program, our group decided not to do any automated tests because we did a lot of beta testing. Beta testing essentially achieves the same goal as automated testing, which is to perform a large number of tests within a small time frame.

7 Trace to Requirements

7.0.1 Load Movies Unit Test (Tests FRAC and FRSE)

This test covers the following requirements (Requirements numbering corresponds to the requirements document): FR1, FR2, FR3

It covers FR1 because when it loads the movies, it parses the data of each movie and prints the movie name to verify that the software is able to provide a synopsis about the movies.

It covers FR2 because it loads movies using the search query passed. The movie names are printed out to show that only movies matching the search query were obtained from the database.

It covers FR3 because no movie names will be printed upon running the test if there are no search hits. This means that no movies will be displayed, which communicates that there were no movies found in the database matching the search query.

7.0.2 Sorting Movies Unit Test (Test FRSR)

This test covers the following requirements (Requirements numbering corresponds to the requirements document): FR5

It covers FR5 because it tests whether the movies have been sorted correctly.

7.0.3 Movie Trailers Unit Test (Test FRTR)

This test covers the following requirements (Requirements numbering corresponds to the requirements document): FR1, FR4, FR6

It covers FR1 because it tests whether the movie data, which includes movie trailers, is loaded successfully into the program.

It covers FR4 because it tests whether the video playback works as expected by playing the videos.

It covers FR6 because it requires the videos to be successfully retrieved from the internet, so that they can be displayed to the user.

8 Trace to Modules

Req.	**Modules**
FRSR	M2, M4, M5, M7
FRSE	M2,M3,M6,M7,M8
FRAC	M5
FRTR	M2,M9, M10, M11, M12

Table 2: Trace Between Functional Requirement Tests and Modules

Req.	**Modules**
SS-1	M1
SS-2	M3,M6
SS-3	M4
SS-4	M5
SS-5	M1
SS-6	M2, M7, M12
SS-7	M3, M7
SS-8	M4, M7

Table 3: Trace Between Non Functional Requirement Tests and Modules

9 Code Coverage Metrics

PGH software solutions has managed to produce roughly a 90% code coverage through our tests. This can be seen through our tables above that trace our tests back to our modules. In these tables, it can be seen that our tests cover all of our modules multiple times meaning that our tests manage to cover a high percentage of our code.