Project Title: System Verification and Validation Plan for Flick Picker

Author Name

November 2, 2022

Revision History

Table 1: Revision History

Date	**Developer(s)**	**Change**
October 31	Jarrod Colwell	Summary & Objectives content added

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations and Acronyms

Symbol	Description
T	Test
V&V	System Verification and Validation
SRS	Software Requirements Specification
UI	User Interface
FR	Functional Requirement
NFR	Non-Functional Requirement
OAuth	Open Authorization

[symbols, abbreviations or acronyms – you can simply reference the SRS (**?**) tables, if appropriate —SS]

This document contains the V&V plan for Flick Picker, as well as the roadmap of when the tests will be implemented.

2 General Information

2.1 Summary

This document describes the plan to verify and validate that Flick Picker meets the defined requirements and specifications. Additionally, this document will validate that Flick Picker fulfills its intended purpose of recommending compatible Movies, TV Shows, or Anime to an individual or a group.

2.1.1 Front End Testing

- Account Creation - Web page that facilitates user account creation
- User Preferences - Web page that facilitates user preference settings
- Group Creation - Web page that facilitates the creation of groups
- Recommendation - Web page that displays recommendations for an individual or group

2.1.2 Back End Testing

- Database Access - Accessing the database to find user preferences or information pertaining to Movies, TV Shows, or Anime
- Recommendation Algorithm - The algorithm responsible for finding the best Movies, TV Shows, or Anime for an individual or group
- API Data - The accessing, storage, and usage of external data from various APIs

2.2 Objectives

2.2.1 Requirements

The first objective involves verifying that Flick Picker meets requirements outlined in our SRS document and validating that the behaviour present is

desirable. This includes the functional requirements (e.g. Authentication Requirements) and the non-functional requirements (e.g. Appearance Requirements). This objective will build confidence in the correctness of Flick Picker along with validating that security and usability standards are met.

2.2.2 UI Elements

The second objective of the V&V document involves the validation of navigability and functionality of UI elements. Each menu described in the 'Front End Testing' section above must be functional for users. Additionally, users must be able to navigate to each page individually. This objective ensures that usability and response time standards are met.

2.2.3 External Connections

The final objective of the V&V document is to ensure that all external connections of Flick Picker (e.g. APIs, Database) work as intended. This objective ensures that the interoperability of Flick Picker is adequate.

2.3 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. —SS]

?

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[You, your classmates and the course instructor. Maybe your supervisor. You shoud do more than list names. You should say what each person's role is for the project. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may just be ad hoc feedback from reviewers, like your classmates, or you may have something more rigorous/systematic in mind.. —SS]

[Remember you have an SRS checklist —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Remember you have MG and MIS checklists —SS]

3.4 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

3.5 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

3.6 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

4 System Test Description

The tests outlined here will follow the following format:

Test #: Test Title

Control: FR: Manual or Automated — NFR: Functional, Dynamic, Manual, or Static

Initial State: Initial State

Input: Input

Output: Output

Derivation: Test Case Derivation

Execution: Automated Selenium Test — API Test

The execution of the test will be through automated selenium tests for the front-end and an API test library for the back-end.

4.1 Tests for Functional Requirements

The tests below are based directly on the FR sections from the SRS. Since these are mandatory requirements, they will all be automated, and the tests must pass on any new code change to prevent these requirements from being violated.

4.1.1 Area of Testing: Authentication

Contains all the tests for the Authentication FR.

UI Auth T 1: User Login - Success

Control: Automated

Initial State: Empty email and password box, on the login screen. Pre-existing test customer exists.

Input: Email and password is filled correctly

Output: Customer is logged in

Derivation: Need a successful UI test on what a customer will see once they are logged in correctly

Execution: Selenium Automated Test

API Auth T 1: User Login - Success

Control: Automated

Initial State: Pre-existing test customer exists.

Input: 'login' API is targeted with correct credentials filled in

Output: Success response is returned (200)

Derivation: Need a successful API test on how the login API will respond to correct information

Execution: API Test

UI Auth T 2: User Login - Failure

Control: Automated

Initial State: Empty email and password box, on the login screen. Pre-existing test customer exists.

Input: Email and password is filled incorrectly

Output: Customer failed to login

Derivation: Need a successful failure UI test on what a customer will see once they input their credentials incorrectly

Execution: Selenium Automated Test

API Auth T 2: User Login - Failure

Control: Automated

Initial State: Pre-existing test customer exists.

Input: 'login' API is targeted with incorrect credentials filled in

Output: Failure response is returned (401)

Derivation: Need a successful API test on how the login API will respond to incorrect information

Execution: API Test

UI Auth T 3: User Sign-Up - Success

Control: Automated

Initial State: Empty email and password box, on the login screen.

Input: Sign-up button is pressed, all boxes for sign-up are filled in correctly, confirm button is pressed

Output: Customer account is created

Derivation: Need a successful UI test on what a customer will see once they Sign-Up correctly

Execution: Selenium Automated Test

API Auth T 3: User Sign-Up - Success

Control: Automated

Initial State: Not Applicable

Input: 'signup' API is targeted with information correctly filled in

Output: Success response is returned (200) and customer account is created

Derivation: Need a successful API test on how the signup API will respond to correct information

Execution: API Test

UI Auth T 4: User Sign-Up - Failure

Control: Automated

Initial State: Empty email and password box, on the login screen.

Input: Sign-up button is pressed, all boxes for sign-up are filled in incorrectly, confirm button is pressed

Output: Areas where there is incorrect information is highlighted

Derivation: Need a successful UI test on what a customer will see if they do not Sign-Up correctly

Execution: Selenium Automated Test

API Auth T 4: User Sign-Up - Failure

Control: Automated

Initial State: Not Applicable

Input: 'signup' API is targeted with information incorrectly filled in

Output: Failure response is returned (422) and customer account is not created

Derivation: Need a successful API test on how the signup API will respond to correct information

Execution: API Test

UI Auth T 5: User Logout

Control: Automated

Initial State: Test customer is logged in, on home page

Input: Logout is pressed

Output: Login screen displays

Derivation: Need a UI test on what a customer will see if they log out

Execution: Selenium Automated Test

API Auth T 5: User Logout

Control: Automated

Initial State: Test customer is logged in

Input: 'logout' API is targeted with the test customer information

Output: Success response is returned (500)

Derivation: Need to test the logout API

Execution: API Test

UI Auth T 6: User OAuth Sign-Up/Login

Control: Automated

Initial State: Empty email and password box, on the login screen.

Input: OAuth button is pressed

Output: Correctly redirect to OAuth login

Derivation: Need a successful UI test on what a customer will see if they choose to sign-up/login though an OAuth provider

Execution: Selenium Automated Test

4.1.2 Area of Testing: Profile/Group

Contains all the tests for the Profile/Group FRs.

UI PG T 1: Modify User Profile

Control: Automated

Initial State: On home page

Input: Profile button is clicked, display name and password are updated, confirm is clicked

Output: Confirmation that information is updated

Derivation: Need a successful UI test on how a customer can update their profile

Execution: Selenium Automated Test

API PG T 1: Modify User Profile

Control: Automated

Initial State: Not Applicable

Input: 'updateProfile' API is targeted with data to update

Output: Success status is returned (200)

Derivation: Need a successful API test on how a customer can update their profile

Execution: API Test

UI PG T 2: Modify User Preferences

Control: Automated

Initial State: On home page

Input: Profile button is clicked, show preferences are updated, confirm is clicked

Output: Confirmation that preferences are updated

Derivation: Need a successful UI test on how a customer can update their preferences

Execution: Selenium Automated Test

API PG T 2: Modify User Preferences

Control: Automated

Initial State: Not Applicable

Input: 'updatePreferences' API is targeted with data to update

Output: Success status is returned (200)

Derivation: Need a successful API test on how a customer can update their preferences

Execution: API Test

UI PG T 3: Create Group

Control: Automated

Initial State: On home page

Input: Create Group button is clicked, name is input, confirm is clicked

Output: Group is created with the specified name

Derivation: Need a successful UI test on how a customer can create a group

Execution: Selenium Automated Test

API PG T 3: Create Group

Control: Automated

Initial State: Not Applicable

Input: 'createGroup' API is targeted with name specified

Output: Success status is returned (200) and group is created

Derivation: Need a successful API test on how a customer can create a group

Execution: API Test

UI PG T 4: Join Group

Control: Automated

Initial State: On home page

Input: Invite is accepted through notifications

Output: User is added to the group

Derivation: Need a successful UI test on how a customer can join a group

Execution: Selenium Automated Test

API PG T 4: Join Group

Control: Automated

Initial State: Not Applicable

Input: 'acceptInvite' API is targeted

Output: Success response is returned (200)

Derivation: Need a successful API test on how a customer can join a group

Execution: API Test

UI PG T 5: Invite to Group

Control: Automated

Initial State: On home page

Input: Group is selected, invite user through their username or email, confirm is clicked

Output: User invite is sent

Derivation: Need a successful UI test on how a customer can send an invite

Execution: Selenium Automated Test

API PG T 5: Invite to Group

Control: Automated

Initial State: Not Applicable

Input: 'sendInvite' API is targeted

Output: Success response is returned (200)

Derivation: Need a successful API test on how a customer can send an invite

Execution: API Test

4.1.3 Area of Testing: Recommendation

Contains all the tests for the Recommendation FRs.

UI R T 1: Receive Recommendations

Control: Automated

Initial State: On home page

Input: Not Applicable

Output: Observe if the home page is updated with recommendations

Derivation: Need a successful UI test where a customer gets rotating recommendations

Execution: Selenium Automated Test

API R T 1: Receive Recommendations

Control: Automated

Initial State: Not Applicable

Input: 'viewRecommendations' API is targeted

Output: Success response is returned (200) along with the list of shows

Derivation: Need a successful API test where a customer gets rotating recommendations

Execution: API Test

UI R T 2: Rate Recommendations

Control: Automated

Initial State: On home page

Input: Hover a movie and input recommendation

Output: Movie should be updated with the recommendation

Derivation: Need a successful UI test where a customer can update recommendations

Execution: Selenium Automated Test

API R T 2: Rate Recommendations

Control: Automated

Initial State: Not Applicable

Input: 'rateRecommendations' API is targeted along with enum data type

Output: Success response is returned (200) and the movie recommendation is stored

Derivation: Need a successful API test where a customer gets updated recommendations

Execution: API Test

API R T 3: Recommendation Match

Control: Automated

Initial State: Not Applicable

Input: 'findSharedRecommendation' API is targeted along with 5 user preferences

Output: Success response is returned (200) and a show is selected with shared preferences

Derivation: Need a successful API test where a group gets a recommended show

Execution: API Test

API R T 4: Recommendation Mismatch

Control: Automated

Initial State: Not Applicable

Input: 'findSharedRecommendation' API is targeted along with 5 user preferences, with entirely different preferences

Output: Success response is returned (200) and a show is selected with shared the highest number of shared preferences

Derivation: Need a successful API test where a group gets a recommended show where none of their preferences matched

Execution: API Test

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. —SS]

[Tests related to usability could include conducting a usability test and survey. —SS]

4.2.1 Area of Testing1

Title for Test

1. test-id1

   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

This section will be filled in after the MIS is completed.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]