System Verification and Validation Plan for Flick Picker

Team 7, 7eam
Talha Asif - asift
Jarrod Colwell - colwellj
Madhi Nagarajan - nagarajm
Andrew Carvalino - carvalia
Ali Tabar - sahraeia

April 6, 2023

Revision History

Table 1: Revision History

Date	**Developer(s)**	**Change**
October 31	Jarrod Colwell	Summary & Objectives content added
October 31	Talha Asif	Added FR Tests
November 1	Ali Tabar	Added and formatted tests for Non-Functional Requirements
November 1	Ali Tabar	Filled in some tests for Non-Functional Requirements (Look and Feel T 1: Appearance, all of Performance)
November 1	Talha Asif	Filled Traceability Matrix
November 2	Andrew Carvalino	NFR tests
November 2	Madhi Nagarajan	Section 3
November 2	Jarrod Colwell	Final Edits
April 5	Andrew Carvalino	Added Survey to Appendix and made final edits

Contents

List of Tables

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

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

SRS: 7eam (2022d)

MG: 7eam (2022b)

MIS: 7eam (2022c)

3 Plan

Contains the plan for Flick Picker, going into detail about the team and verification plans

3.1 Verification and Validation Team

The Verification and Validation Team will consist of Internal (team) and External members.

1. Team Members:

(a) Ali: Ad-hoc or checklist verification of SRS, Design documents through peer-reviewing. Verification of implementation. For testing, responsible for creating/running frontend unit tests and system tests (Integration, Manual).

(b) Andrew: Ad-hoc or checklist verification of SRS, Design documents through peer-reviewing. Verification of implementation. For testing, responsible for creating/running frontend unit tests and system tests.

(c) Jarrod: Ad-hoc or checklist verification of SRS, Design documents through peer-reviewing. Verification of implementation. For testing, responsible for creating/running backend unit tests and system tests.

(d) Madhi: Ad-hoc or checklist verification of SRS, Design documents through peer-reviewing. Verification of implementation. For testing, responsible for creating/running frontend & backend unit tests and system tests.

(e) Talha: Ad-hoc or checklist verification of SRS, Design documents through peer-reviewing. Verification of implementation. For testing, responsible for creating/running frontend & backend unit tests and system tests.

2. External Members

(a) Classmates: Ad-hoc verification of SRS and Design documents.

(b) Course TAs and Instructors: Ad-hoc verification of SRS and Design documents. Ad-hoc verification of the app implementation.

(c) Users: Ad-hoc validation of application.

3.2 SRS Verification Plan

For the SRS Verification Plan, an ad-hoc approach will be followed. Teammates will peer-review work done for the SRS to verify whether both the functional and non-functional requirements serve validity. The SRS checklist will be followed prior to the finalization of a revision by a team member. Classmates and instructors will ad-hoc review the Revision 0 of the SRS, again to verify the requirements.

3.3 Design Verification Plan

For the Design Verification Plan, an ad-hoc approach will be followed. Teammates will peer-review work done for the MG and MIS to verify whether both modules and their specification have validity. The MG and MIS checklists will be followed prior to the finalization of a revision by a team member. Classmates and instructors will ad-hoc review the Revision 0 of the MG and MIS, again to verify the modules.

3.4 Implementation Verification Plan

Our Implementation Verification Plan will mainly consist of system testing and unit testing, as specified in the latter sections of this document. These tests will ensure that our implementation meets the criteria for our functional and non-functional requirements. Further means of implementation verification will also include code walkthroughs and reviews prior to pull request approvals. Static analysis and code coverage will also be incorporated to ensure that new code is written up-to standards. Verification tools will be explained in the latter sections.

3.5 Automated Testing and Verification Tools

Unit tests will be written with a JavaScript unit testing framework called Jest. Jest also produces reports for code coverage metrics. Our code coverage goals include an overall 80% coverage. A code coverage check will be done for every Pull Request made. Code coverage reports will be monitored to ensure that all functionality is being tested. Integration/system tests will be done with Selenium and Cucumber.js. Cucumber.js also has functionality to produce testing metrics and results. For static analysis and linting, ESLint and AirBnB styling will be utilized. This will ensure uniform coding standards and uncover any potential errors prior to compilation. Auomated code builds, through GitHub/Jenkins, will be present to ensure that every deployment made will not break the application. Refer to Development Plan for indepth details behind the tools:

7eam (2022a)

3.6 Software Validation Plan

For software validation, our team will host review sessions with target users of our application. During this session, these users will interact with the application and we will record their experiences. Based on the data collected, it will allow us to validate whether our requirements fulfills the goal of our application.

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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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: Manual 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 tests below are based directly on the NFR sections from the SRS. Not all tests will be automated due to the nature of the requirement being tested. We have also deemed to not test Scalability, due to the nature of that requirement.

4.2.1 Area of Testing: Look and Feel

Contains all the tests for the Look and Feel NFR.

Look and Feel T 1: Appearance

Control: Manual

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

Input: Any and all appropriate user inputs and navigation (typing login info, clicking buttons)

Output: Visiting all menus and triggering all possible UI changes

Derivation: Program's visual look in the real world (proper buttons / menus appearing, alignment, etc) should be approved by a real user

Execution: Manual User Test

Look and Feel T 2: Ease of Use

Control: Non-Functional

Initial State: System is in the login screen state

Input: User logs in and explores the different screens and services offered

Output: User goes through the system and provide feedback on the style choices (such as size of buttons or overall colour palette)

Derivation: The test is meant to assess how different users may feel about style decisions and offer feedback on what looks and feels good to them and what needs improvement

Execution: Survey

4.2.2 Area of Testing: Usability and Humanity Tests

Usability and Humanity T 1: Ease of Use

Control: Non-Functional

Initial State: System is in the login screen state

Input: User logs in and explores the different screens and services offered

Output: User goes through the system without needing to ask for assistence or misunderstanding the affordances of the different screens

Derivation: The test case is meant to assess whether the presentation of the system informs users of different technological skill levels of what functionality is provided by whatever state/screen they may be on

Execution: User Test

Usability and Humanity T 2: Learning

Control: Non-Functional

Initial State: Users have just finished Test #3

Input: Users are given a survey of how easy/complicated their experience using the system was

Output: The surveys are filled out and collected

Derivation: This survey is a continuation of Test #3, with its results providing feedback as to whether the GUI of the system is understandable to users of different skill levels

Execution: Survey

4.2.3 Area of Testing: Performance

Contains all the tests for the Performance NFR.

Performance T 1: User Input Responsiveness

Control: Dynamic

Initial State: Any menu

Input: Any user input (entering text, clicking a button)

Output: Appropriate program response to user input

Derivation: Must time the responsiveness of the program to an arbitrary user action in order to gauge if it is below 1 second.

Execution: Manual Test

Performance T 2: Safety-Critical

Control: Dynamic

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

Input: Any and all appropriate user inputs and navigation (typing login info, clicking buttons)

Output: Visiting all menus and triggering all possible UI changes, verifying with Selenium that no user information is outputted on the page

Derivation: Ensuring that the application handles the user's private data properly and does not output it due to any unforeseen condition

Execution: Manual Test

Performance T 3: Precision or Accuracy

Control: Manual

Initial State: Main screen to generate a recommended media match for a previously created and inputted group

Input: Clicking the button to generate the match

Output: The media match

Derivation: Must assess accuracy of program's ability to find something that everyone in the group should enjoy

Execution: Conduct a survey using a fixed amount of people, and have each of them create an account. An effort should be made to choose a group of respondents that have a wide variety of preferences for the different categories of media available on Flick Picker. They will create and join multiple groups among each other using the application, and search for matches in each of these distinct groups. The survey will inquire respondents about each group, and how accurate they perceive their matches were each given group.

Performance T 4: Reliability and Availability

Control: Dynamic

Initial State: Online application with server running

Input: Visiting the web page

Output: Connection success or failure

Derivation: Checking to see if the web application is available for at least 90% of the day, assuming Firebase stability

Execution: Manual Test

Performance T 5: Robustness or Fault-Tolerance

Control: Dynamic

Initial State: Main screen to generate a recommended media match for a previously created and inputted group. This group will be comprised of unique and few preferences.

Input: Input

Output: Output

Derivation: Verifying that a match will always be found by stress testing using some heterogeneous groups

Execution: Manual Test

Performance T 6: Capacity

Control: Dynamic

Initial State: 0 users logged on the application

Input: Several users logging onto the application

Output: Application's stability

Derivation: Ensuring and proving that each user session is independent of another and that no limit to the number of users logged on concurrently exists.

Execution: Manual Test

4.2.4 Area of Testing: Operational and Environmental Test

Operational and Environmental T 1: Interfacing with Adjacent Systems

Control: Manual

Initial State: System is in the login menu state

Input: Developer will use the system as a regular user would on different web browsers

Output: The system will have the same fundamental functionalities on all browsers, with perhaps minor differences in visual presentation

Derivation: Developer will go through the various functionalities of the system on different web browsers, such as Chrome and Firefox, to assess whether or not it works and is usable on each

Execution: Developer Exploration Test

4.2.5 Area of Testing: Maintainability and Support Tests

Maintainability and Support T 1: Adaptability

Control: Manual

Initial State: System is in the login menu state

Input: Developer will use the system as a regular user would on different mobile devices

Output: The system will have the same fundamental functionalities on different mobile devices, with perhaps minor differences in visual presentation

Derivation: Developer will go through the various functionalities of the system to assess whether or not it works and is usable on each

Execution: Developer Exploration Test

4.2.6 Area of Testing: Security Tests

Security T 1: Access

Control: Manual

Initial State: System is in the main menu state

Input: Developer will click on their user profile and test the ability to change settings, as well as view the profile information of other users and ensure they cannot change their settings

Output: The Developer's user profile will be changed, with such changes being visible to other users (if the setting was one which can be viewed by others) with the content of other profiles being unchangeable

Derivation: Developer will ensure that the user's data is changable, that changes to public information can be viewed by others, that others' settings cannot be changed, and private information cannot be viewed by other users

Execution: Developer Exploration Test

Security T 2: Integrity

Control: Manual

Initial State: System is in the main menu state

Input: Developer will change a profile or preference setting

Output: After different increments of time after the changes are made (such as immediately, 1 hour, and 24 hours after) the Developer will verify that all the changes made persist and do not revert to a prior state

Derivation: This is the best way to verify that information persists and retains its integrity over time, with it being possible to infer future persistence of information after the testing period

Execution: Developer Exploration Test

4.3 Traceability Between Test Cases and Requirements

The two traceability matrices can be found in the Appendix.

5 Unit Test Description

In creating unit tests, we wanted to prioritize those functions of the system that were most important in order that it may work. This meant creating tests that covered the essentials and, in some cases, could be mapped onto similar functions (such as the group voting session testing results being applicable to individual voting, thereby eliminating the need to test the latter). Tests whose results followed from those of others were not included for the sake of brevity.

5.1 Unit Testing Scope

M3, M5, M6, and M8 is the scope of the modules tested with unit testing. Each of these modules had functions that were vital to the system as a whole, and so they were prioritized for unit testing to speed up the testing process whenever large updates to the source code were made. We chose to leave out M4 (the Friends Module) since the creation of new test users and adding them as friends was a feature that could easily be verified manually and, if something was wrong with it, would be covered by the tests for the Groups Module. Similarly, M9 (OAuth Module) was left out of the scope since its functionality relied on what is provided by Google and Meta. Finally, M10 (API Module) was also left out of the scope since it, too, was dependent on what is provided by the API developers.

5.2 Tests for Functional Requirements

5.2.1 Module 3 - Native Login

The most important function of this module is the creation of new accounts, with their data being uploaded to the database, thereby making the functionality of logging in inferrable from the success of this unit test.

T1: User Creation

Type: Automatic

Initial State: Login Page

Input: Test email and password

Output: New user profile created and added to database

Test Case Derivation: Ensure that the user creation funcationality works

How test will be performed: automatic testing file (user.test.ts)

5.2.2 Module 5 - Groups

The funcationality of the Groups Module was fully covered by testing the creation of a group and all funcationality surrounding the adding of new members.

T2: Create a group

Type: Automatic

Initial State: Create Group Page

Input: Test group name

Output: New group is created

Test Case Derivation: Ensure that users can successfully create new groups

How test will be performed: automatic testing file (group.test.ts)

T3: Add user to group

Type: Automatic

Initial State: Social Page

Input: Group name and user id/email

Output: varification that selected user has been added to group

Test Case Derivation: Ensure that users can successfully be added to groups

How test will be performed: automatic testing file (group.test.ts)

T4: Invite a user to group

Type: Automatic

Initial State: Social Page

Input: Group name and friend id/email

Output: New user recieves invite to test group

Test Case Derivation: Ensure that users can successfully invite new members to groups

How test will be performed: automatic testing file (group.test.ts)

T5: Accept Invite to group

Type: Automatic

Initial State: Join Group Page

Input: Group invite id

Output: User accepts the group invite

Test Case Derivation: Ensure that users can successfully accept invitations to new groups

How test will be performed: automatic testing file (group.test.ts)

5.2.3 Module 6 - Profile

The changes to the profile section all used similar functionality with Firebase, so we thought it only necessary to test the creation of a new username. Moreover, the preferenes were fully tested, with initial and later preference changes being accounted for.

T6: Edit user profile

Type: Automatic

Initial State: Change User Name Page

Input: New user profile name

Output: Newly entered profile name is saved and displays instead of email or previous profile name

Test Case Derivation: Ensure that the user can successfully change their profile name

How test will be performed: automatic testing file (user.test.ts)

T7: New preferences for a user

Type: Automatic

Initial State: Preferences Page

Input: Set of media preferences

Output: Creation of the user's first set of preferences

Test Case Derivation: Ensure that the user can successfully create their initial preferences

How test will be performed: automatic testing file (user.test.ts)

T8: Edit user preferences

Type: Automatic

Initial State: Preferences Page

Input: Set of media preferences

Output: Change the user's first set of preferences to the newly specified preferences

Test Case Derivation: Ensure that the user can successfully change their preferences

How test will be performed: automatic testing file (user.test.ts)

5.2.4 Module 8 - Matching Algorithm

The matching algorithm was fully tested, with the voting session, recommendations, votes, and final suggestion all being accounted for.

T9: Start a Voting Session

Type: Automatic

Initial State: Group Page

Input: Group id

Output: New voting session for test group

Test Case Derivation: Ensure that the group can start a new voting session

How test will be performed: automatic testing file (recommendation.test.ts)

T10: Load Recommendations

Type: Automatic

Initial State: Group Voting Page

Input: Group and session id

Output: Recommendations for the group

Test Case Derivation: Ensure that the algorithm can generate recommendations based on the preferences of all members

How test will be performed: automatic testing file (recommendation.test.ts)

T11: Submit Votes

Type: Automatic

Initial State: Group Voting Page

Input: Session id and votes for suggested media

Output: Verification that the users' votes are saved

Test Case Derivation: Ensure that the system can save users' votes for different media

How test will be performed: automatic testing file (recommendation.test.ts)

T12: Find Best Match

Type: Automatic

Initial State: Group Voting Page

Input: Session id

Output: The media that the algorithm chooses based on preferences and votes

Test Case Derivation: Ensure that the algorithm can provide a decision based on the suggestions and votes

How test will be performed: automatic testing file (recommendation.test.ts)

5.3 Tests for Nonfunctional Requirements

There were no nonfunctional unit tests, as all nonfunctional requirements were tested through different means (such as the survey, as seen in the Appendix).

5.4 Traceability Between Test Cases and Modules

Table 2: Module Traceability Matrix

Test Number	**Module**
T1: User Creation	M3
T2: Create a group	M5
T3: Add user to group	M5
T4: Invite a user to group	M5
T6: Edit user profile	M6
T7: New preferences for a user	M6
T8: Edit user preferences	M6
T9: Start a Voting Session	M8
T10: Load Recommendations	M8
T11: Submit Votes	M8
T12: Find Best Match	M8

References

7eam. Development plan. https://github.com/TAsif/Team7Project/blob/develop/docs/DevelopmentPlan/DevelopmentPlan.tex, 2022a.

7eam. Module guide. https://github.com/TAsif/Team7Project/blob/develop/docs/Design/MG/MG.tex, 2022b.

7eam. Module interface specification. https://github.com/TAsif/Team7Project/blob/develop/docs/Design/MIS/MIS.tex, 2022c.

7eam. System requirements specification. https://github.com/TAsif/Team7Project/blob/develop/docs/SRS/SRS.tex, 2022d.

Appendix

Table 3: Traceability Matrix 1

Test Number	**Requirement**
UI Auth T 1	FR 1
API Auth T 1	FR 1
UI Auth T 2	FR 1
API Auth T 2	FR 1
UI Auth T 3	FR 1
API Auth T 3	FR 1
UI Auth T 4	FR 1
API Auth T 4	FR 1
UI Auth T 5	FR 3
API Auth T 5	FR 3
UI Auth T 6	FR 2
UI PG T 1	FR 4
API PG T 1	FR 4
UI PG T 2	FR 5
API PG T 2	FR 5
UI PG T 3	FR 6
API PG T 3	FR 6
UI PG T 4	FR 7
API PG T 4	FR 7
UI PG T 5	FR 8
API PG T 5	FR 8
UI R T 1	FR 9
API R T 1	FR 9
UI R T 2	FR 10
API R T 2	FR 10
UI R T 3	FR 11
API R T 3	FR 11

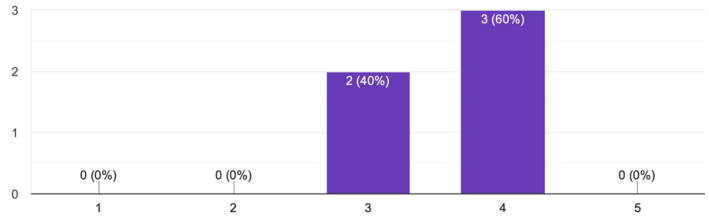
Table 4: Traceability Matrix 2

Test Number	**Requirement**
Look and Feel T 1	10.1.1
Look and Feel T 2	10.1.2
Usability and Humanity T 1	10.2.1
Usability and Humanity T 2	10.2.3
Performance T 1	10.3.1
Performance T 2	10.3.2
Performance T 3	10.3.3
Performance T 4	10.3.4
Performance T 5	10.3.5
Performance T 6	10.3.6
Operational and Environmental T 1	10.4.3
Maintainability and Support T 1	10.5.3
Security T 1	10.6.1
Security T 2	10.6.2

User Survey

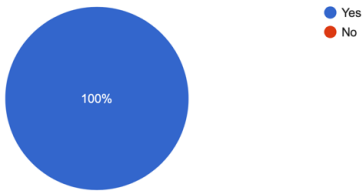
How technologically proficient are you, with 1 being not at all and 5 being very? (optional)

5 responses



Were you able to navigate through all the different screens (ie. no features were inaccessible)?

5 responses



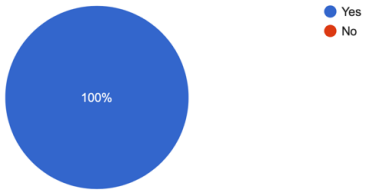
Were there any buttons or features that didn't work how you thought they would?

5 responses

- Seems easy enough for any user to navigate. Every button was self explanatory.
- No
- I thought the "seen it before" button would automatically deselect itself
- Everything worked as expected and the pages were easy to navigate.
- Yes

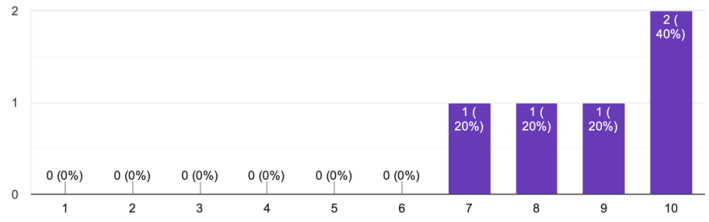
Were you able to go through the app's screens and inputs without help from someone else?

5 responses



How would you rate how easy/hard it was to understand the app and its functions, with 1 being very complicated and 10 being completely understandable?

5 responses



How would you rate the colour scheme and layout of the user interface, with 1 being completely inconsistent and 10 being perfectly consistent?

5 responses