**Test plan for**

**Watchlist+**

*ChangeLog*

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Change Date** | **By** | **Description** |
| 1 | 17/02/2023 | Tyler Clark | Filled in initial info |
| 2 | 18/03/2023 | Tyler Clark | Changed some planned tests and add test environment for integration tests |
| 3 | 19/03/2023 | Tyler Clark | Wrote acceptance tests for each feature |
| 4 | 16/04/2023 | Tyler Clark | Update with final tests for 100% coverage |

# Introduction

## Scope

**In Scope:**

* **Watchlist**
* **View & Browse**
* **Reviews & Recommendations**
* **Account Management**

**Out of Scope:**

* **More detailed title view information**
* **More detailed actor information**
* **More detailed director & producer information**
* **Updating DB with new titles**

## Roles and Responsibilities

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Net ID** | **GitHub username** | **Role** |
| Tyler Clark | clarkt35 | Br0k3N-ET3rNaL | All |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Test Methodology

## Test Levels

* **Account Management:**
  + Unit Tests 1-5: Backend user data validation (user.service.test.js)
  + Unit Tests 6-13: Frontend user data validation (log-in.test.tsx & sign-up.test.tsx)
  + Integration Test 1: Test creating a new user
  + Integration Test 2: Test verifying an existing user
  + Integration Test 3: Test checking if email exists
  + Integration Test 4: Test checking if username exists
  + Acceptance Critieria: User can create an account and sign-in
  + Acceptance Test:
    - Steps:
      * Open the application
      * Click sign-up in the top right
      * Enter username below “Username:”
      * Enter email below “Email:”
      * Enter password below “Password:”
      * Enter the same password again below “Repeat Password:”
      * Click button labeled “Sign Up”
      * Automatically re-directed to login screen
      * Enter email used to sign-up below “Email:”
      * Enter password used to sign-up below “Password:”
    - Expected Result: User has an account and is now logged-in
* **View & Browse:**
  + Unit Tests 1 & 2: Backend; check valid sorting columns (title.repository.test.js)
  + Unit Test 3: Browse view; test sorting functionality (browse-view.test.tsx)
  + Unit Test 4: Browse view; test page changing functionality (browse-view.test.tsx)
  + Unit Test 5: Browse view; test search functionality (browse-view.test.tsx)
  + Unit Test 6: Combination of previous 3 unit tests (browse-view.test.tsx)
  + Unit Test 7: Title list element; test display of data (title-list-element.test.tsx)
  + Unit Test 8: Title list element; test click functionality (title-list-element.test.tsx)
  + Unit Test 9: Title view; test display of data (title-view.test.tsx)
  + Unit Test 10: Title view; test closing of window (title-view.test.tsx)
  + Unit Test 11: Browse view; test opening and closing title view (browse-view.test.tsx)
  + Unit Test 12: Browse view; test opening and closing edit view (browse-view.test.tsx)
  + Unit Test 13: Browse view; test opening and closing edit view while logged in (browse-view.test.tsx)
  + Integration Test 5: Test getting a page of titles
  + Acceptance Critieria: User can scroll through the browse view, switch pages, search for specific titles, and change sorting options
  + View Acceptance Test:
    - Steps:
      * Open the application
      * Click on the title of a displayed movie/show
    - Expected Result: User can see a description of the selected title
  + Search Acceptance Test:
    - Steps:
      * Open the application
      * Enter desired search in textbox at the top of the screen
    - Expected Result: User can now view titles that match the entered search
  + Sort Acceptance Test:
    - Steps:
      * Open the application
      * Select desired sorting by clicking on the dropdown below “Sort By” and clicking the desire sorting option
    - Expected Result: User can now view titles sorted by the desired sorting option
  + Page Acceptance Test:
    - Steps:
      * Open the application
      * Scroll to the bottom of the page
      * Click the button labeled “>”
    - Expected Result: User can now view the next page of titles
* **Reviews & Recommendations:**
  + Unit Test 1: View reviews; test display of single review
  + Unit Test 2: View reviews; test display of list of reviews
  + Unit Test 3: Leave review; test creation of new review
  + Unit Test 4: View review; test display of no reviews
  + Unit Test 5: Test close review view
  + Unit Test 6: Test close create review view
  + Unit Test 7: View recommendations; test display of single recommendations
  + Uint Test 8: View recommendations; test display of list of recommendations
  + Unit Test 9: Recommend; test creation of recommendation
  + Unit Test 10: Recommend; test creation of recommendation where title is not unique
  + Unit Test 11: Test close recommendations view
  + Unit Test 12: Test close create recommendation view
  + Unit Test 13: View reviews; test switching pages
  + Unit Test 14: View recommendations; test switching pages
  + Unit Test 15: Test opening and closing review view from title view
  + Unit Test 16: Test opening and closing recommendations view from title view
  + Unit Test 17: Test opening and closing create review from title view
  + Unit Test 18: Test opening and closing create recommendation from title view
  + Integration Test 6: Test creating reviews
  + Integration Test 7: Test getting a page of reviews
  + Integration Test 8: Test creating recommendations
  + Integration Test 9: Test getting a page of recommendations
  + Integration Tests 10 & 11: Test checking title exists when creating a recommendation
  + Acceptance Critieria: User can leave reviews for titles they’ve seen, view other users’ reviews, leave recommendations for titles they’ve seen, and view other users’ recommendations
  + Create Review Acceptance Test:
    - Pre-conditions:
      * User is logged-in
      * User has selected a title
    - Steps:
      * Click the button labeled “Leave a Review”
      * Enter review in textbox
      * Click button labeled “Submit”
    - Expected Result: User has left a review for the selected title
  + View Reviews Acceptance Test:
    - Pre-conditions:
      * User has selected a title
    - Steps:
      * Click the button labeled “View Reviews”
    - Expected Result: If there are any reviews for the selected title, they will be visible to the user, otherwise the user will see “No reviews yet”
  + Create Recommendation Acceptance Test:
    - Pre-conditions:
      * User is logged-in
      * User has selected a title
    - Steps:
      * Click the button labeled “Recommend Another Title”
      * Enter the title of the movie/show to recommend based on the selected title
      * Click the button labeled “Submit”
      * If there are multiple movies/shows with the entered title, then
        + Enter movie or show below “Movie/Show”, depending on the type of the title
        + Enter the release year of the movie/show below “Release Year”
        + Click the button labeled “Submit”
    - Expected Result: The user has left a recommendation for the entered title, based on the selected title
  + View Recommendations Acceptance Test:
    - Pre-conditions:
      * User has selected a title
    - Steps:
      * Click the button labeled “View Recommendations”
    - Expected Result: If there are any recommendations for the selected title, they will be visible to the user, otherwise the user will see “No recommendations yet”
* **Watchlist:**
  + Unit Test 1: Test adding title to watchlist
  + Unit Test 2: Test removing title from watchlist
  + Unit Test 3: Test switching between watchlist views
  + Unit Test 4: Test display watchlist element
  + Unit Test 5: Test watchlist sorting functionality
  + Unit Test 6: Test watchlist links to display title and edit
  + Unit Test 7: Test basic watchlist display
  + Unit Test 8: Test close edit watchlist view
  + Unit Test 9-10: Test editing watchlist items
  + Unit Test 11: Test removing title from watchlist from browse view
  + Unit Test 12: Test switching pages in watchlist
  + Unit Test 13: Test opening and closing title view from watchlist
  + Integration Test 12: Test updating a watchlist entry
  + Integration Test 13: Test deleting a watchlist entry
  + Integration Test 14: Test adding title to watchlist
  + Integration Test 15: Test getting a page of watchlist entries
  + Acceptance Critieria: User can view their watchlist, add titles to it, remove titles from it, switch between watchlist views, edit items in their watchlist, sort their watchlist, rate titles on their watchlist
  + Add to Watchlist Acceptance Test:
    - Pre-conditions:
      * User is logged-in
    - Steps:
      * Click the button labeled “add” next to the title to add
    - Expected Result: The selected title has been added to the user’s watchlist
  + View Watchlist Acceptance Test:
    - Pre-conditions:
      * User is logged-in
    - Steps:
      * Click “Watchlist” in the top right
    - Expected Result: User can view their watchlist
  + Sort Watchlist Acceptance Test:
    - Pre-conditions:
      * User is logged in
      * User is viewing their watchlist
    - Steps:
      * Select desired sorting by clicking on the dropdown below “Sort By” and clicking the desire sorting option
    - Expected Result: User can now view their watchlist sorted by the desired sorting option
  + Watchlist Page Acceptance Test:
    - Pre-conditions:
      * User is logged in
      * User is viewing their watchlist
    - Steps:
      * Scroll to the bottom of the page
      * Click the button labeled “>”
    - Expected Result: User can now view the next page of their watchlist
  + Filter Watchlist Acceptance Test:
    - Pre-conditions:
      * User is logged in
      * User is viewing their watchlist
    - Steps:
      * Select desired status filter by clicking the dropdown below “Status” and clicking the desired status filter
    - Expected Result: User can now view their watchlist filtered based on the selected status filter
* Regression testing covering all previously mentioned unit and integration tests to be done automatically on commit using GitHub Actions

## Test Completeness

* 100% code coverage
* All Manual & Automated Test cases executed
* No flaky tests
* All open bugs are fixed or will be fixed in next release

# Resource & Environment Needs

## Testing Tools

* **Bug & Requirement Tracking:** GitHub
* **Automation:** GitHub Actions

## Test Environment

1. Windows 8 and above
2. GitHub Actions
3. Jest + ts-jest + testing-library for Frontend
4. Jest for Backend
5. Jest + supertest for integration tests

# Terms/Acronyms

Make a mention of any terms or acronyms used in the project

| TERM/ACRONYM | DEFINITION |
| --- | --- |
| API | Application Program Interface |
| AUT | Application Under Test |