TEST PLAN FOR <<BOOKWYRM>>

ChangeLog

Version	Change Date	By	Description
version number	Date of Change	Name of person who made changes	Description of the changes made
1.0.0	2022-03-03	Luke Morrow	Initial setup, Added backend unit tests
1.0.1	2022-03-05	Gurtej Boparai	Added frontend unit tests
1.0.2	2022-03-28	Cameron Jung	Updated reviewComponent tests for using local storage
1.0.3	2022-03-31	Cameron Jung	Added tests for RatingComponent and StarRatingWrapperComponent
1.1	2022-04-03	Luke Morrow	Update existing tests to match what is found in the project. Add acceptance tests. Update coverage goal to be more specific
1.1.1	2022-04-03	Luke Morrow	Update acceptance test clarity
1.2	2022-04-22	Luke Morrow	Added Integration Tests
1.2.1	2022-04-24	Luke Morrow	Update acceptance tests

1 Introduction

This document contains the testing plan for the BookWyrm project in which the following will be described: scope, testing methodology, resources and environment requirements, and terms used.

1.1 Scope

The scope of our testing will include front end(Vue) and back end(Spring) unit tests focusing on the logic that happens around the API seam. More specifically, the unit tests will focus on the Vue components files and the Spring Controller files.

1.2 Roles and Responsibilities

Detailed description of the Roles and responsibilities of different team members like. Note you only need to list the role you have in your team. There are some example roles.

- QA Analyst
- Test Manager
- Configuration Manager
- Developers
- Installation Team

Amongst others.

Name	**Net ID**	**GitHub username**	**Role**
Luke Morrow	morrowl4	LukeBMorrow	Test Manager
Cameron Jung	jungc	CameronJung	Developer
Gurtej Boparai	boparai3	gurtejboparai	Developer
Long Vu	vuml	louismacvux	Developer
Antony Anuraj	anuraja	antonyanuraj	Developer

2 Test Methodology

2.1 Test Levels

Test Levels define the Types of Testing to be executed on the Application Under Test (AUT). In this course, **unit testing, integration testing, acceptance testing, regression testing, and load testing** are mandatory. Please describe how you will do these tests. You may skip load testing at this moment. Please revisit it after the related lecture is given.

Requirements:

- List the class/method/core feature you plan to test and how you would like test them and its acceptance criteria.
- For unit testing, at least 10 unit tests for each core feature to cover the code related to each core feature
- For integration testing, at least 10 in total to cover core features.
- Acceptance testing for each core feature (if it is manual, need to list the steps)
- For regression testing, you need to execute all above unit tests + integration tests you have for each commit pushed to the main branch.

Unit Tests

Backend Tests

Feature	Books
Class	BookValidator
Method	validateUploadInformation

Test 1: Happy Path
Input: valid BookUploadInput object
Expected output: an empty ArrayList

Test 2: Missing author
Input: BookUploadInput object with title but no author
Expected output: an ArrayList with a single author missing error

Test 3: Missing title
Input: BookUploadInput object with author but no title
Expected output: an ArrayList with a single title missing error

Test 4: Missing everything
Input: BookUploadInput object with neither a title or an author
Expected output: an ArrayList with a both title missing and author missing errors

Feature	Books
Class	BookValidator
Method	validateUpdateInformation

Test 1: Happy Path
Input: valid BookUpdateInput object
Expected output: an empty ArrayList

Test 2: Missing desc
Input: BookUpdateInput object with no desc
Expected output: an ArrayList with a single error

Test 3: Missing id
Input: BookUpdateInput object with no id
Expected output: an ArrayList with a single error

Test 4: Missing everything
Input: BookUpdateInput object with every required field missing
Expected output: an ArrayList with one error per missing field

Feature	Reviews
Class	ReviewValidator
Method	validateUploadInformation

Test 1: Happy Path
Input: valid ReviewUploadInput object
Expected output: an empty ArrayList

Test 2: Missing author
Input: ReviewUploadInput object missing an author
Expected output: an ArrayList with a single author missing error

Test 3: Missing book Id
Input: ReviewUploadInput object missing a book Id
Expected output: an ArrayList with a single id missing error

Test 4: Missing AnonymousFlag
Input: ReviewUploadInput object with missing AnonymousFlag

Expected output: an empty ArrayList

Test 5: Empty Ratings
Input: ReviewUploadInput object missing ratings
Expected output: an ArrayList with single error

Test 6: Missing Ratings
Input: ReviewUploadInput object missing ratings
Expected output: an ArrayList with single error

Test 7: Missing Ratings
Input: ReviewUploadInput object missing ratings
Expected output: an ArrayList with single error

Test 8: Missing everything
Input: ReviewUploadInput object missing every field
Expected output: an ArrayList with previously indicated errors

Test 9: Good voting input
Input: ReviewVotingInput object valid
Expected output: an ArrayList with no errors

Test 10: Missing Review Id
Input: ReviewVotingInput object missing review Id
Expected output: an ArrayList with one error

Test 11: Missing User Id
Input: ReviewVotingInput object missing user Id
Expected output: an ArrayList with one error

Test 12: Missing vote value
Input: ReviewVotingInput object missing vote value
Expected output: an ArrayList with one error

Test 13: Good voting input
Input: ReviewVotingInput object valid
Expected output: an ArrayList with one error for each previous voting input test

Feature	Comments
Class	CommentValidator
Method	validateCommentInformation

Test 1: Happy Path

Input: valid CommentUploadInput object
Expected output: an empty ArrayList

Test 2: Missing author
Input: CommentUploadInput object with missing author
Expected output: an ArrayList with a single author missing error

Test 3: Missing reviewId
Input: CommentUploadInput object with missing reviewId
Expected output: an ArrayList with a single Id missing error

Test 4: Missing Content
Input: CommentUploadInput object with missing Content
Expected output: an ArrayList with a single Id missing error

Test 5: Missing AnonymousFlag
Input: CommentUploadInput object with missing AnonymousFlag
Expected output: an empty ArrayList

Test 6: Missing everything
Input: CommentUploadInput object with neither a title or an author
Expected output: an ArrayList with all the previously expected errors

Frontend Tests

Feature	Book
Component	BookBriefView

Test 1: renders the component
Test 2: processes valid prop data

Feature	Comment
Component	CommentComponent

Test 1: renders the component
Test 2: processes valid prop data

Feature	Review
Component	ReviewComponent

Test 1: renders the component
Test 2: processes valid prop data

Feature	Book
Component	SearchBar

Test 1: renders the component
Test 2: processes valid prop data

Feature	Review + Comment
Component	userComponent

Test 1: renders the component
Test 2: processes valid prop data

Integration Tests

Feature	Books
Class	BookController
Method	createBook

Test 1: Happy Path
Input: valid data
Expected output: 200 status with body containing name from data

Test 2: Bad Request
Input: invalid data (Missing either title or author)
Expected output: 400 status with a non-empty array in errorList

Feature	Books
Class	BookController
Method	searchBookByTitle

Test 1: testGoodSearch (Search for book by title)

Input: name of book in DB
Expected output: 200 status with body containing name from data

Test 2: testEmptySearch (Search for book by title)
Input: name of book not in DB
Expected output: 404 status with body containing error message

Feature	Books
Class	BookController
Method	searchBookById

Test 1: testGoodSearchById (Search for book by book ID)
Input: valid data
Expected output: 200 status with body containing name from data

Test 1: testEmptyIdSearch (Search for book by book ID)
Input: non-existent Id
Expected output: 404 status with body containing error message

Feature	Books
Class	BookController
Method	updateBookDesc

Test 1: testGoodUpdate
Input: valid data
Expected output: 200 status with body containing name from data

Test 1: testBadUpdate
Input: non-existent Id
Expected output: 400 status with body containing error message

Feature	Comments
Class	CommentController
Method	createComment

Test 1: Happy Path

Input: valid data
Expected output: 200 status with body containing name from data

Test 2: Bad Request
Input: invalid data (Missing either title, author, or anonymous flag)
Expected output: 400 status with a non-empty array in errorList

Feature	Reviews
Class	ReviewController
Method	createReview

Test 1: Happy Path
Input: valid data
Expected output: 200 status with body containing name from data

Test 2: Bad Request
Input: invalid data (Missing either title, author, or Id)
Expected output: 400 status with a non-empty array in errorList

Feature	User Roles
Class	UserController
Method	signup

Test 1: testHappyPathSignup
Input: valid data
Expected output: 200 status with no error messages in body

Test 2: testBadRequest
Input: invalid data
Expected output: 400 status with a non-empty array in errorList

Feature	User Roles
Class	UserController
Method	signin

Test 1: testHappyPathSignIn
Input: valid data

Expected output: 200 status with body containing name from data

Test 2: testFailedSignIn
Input: invalid data (Missing either title, author, or Id)
Expected output: 400 status with a non-empty array in errorList

Feature	User Roles
Class	UserController
Method	updateUser

Test 1: testHappyPathUpdate
Input: valid data
Expected output: 200 status with body containing updated user info

Test 2: testFailedUpdate
Input: invalid data
Expected output: 400 status with a non-empty array in errorList

Acceptance Tests

These tests are performed assuming the docker container is running and hosting the app on http://localhost:80/ .

1. As a **user**, I need to be able to search for books.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Enter the name of the book to search in the “Search” box and hit the “Go” button.
    - If the book exists, it will appear in the list
2. As a **user**, I need to be able to submit new books that have not been reviewed before.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Enter the name of the book to search in the “Search” box and hit the “Go” button.
    - If the book is not found, click on the “create” button.
    - Enter the title, author and description of the book into their respective forms and hit the “Submit” button.
    - The page will redirect to the newly created book’s detail page

3. As a **user**, I need to create reviews for a book.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Enter the name of the book to review in the "Search" box and hit the "Go" button.
    - Click on the appeared book.
    - Click on "Add a review".
    - Fill the review form and hit the 'Submit" button.
    - The new review will appear at the bottom of the list.
4. As a **user**, I need to read reviews for a book.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Enter the name of the book to read the reviews for in the "Search" box and hit the "Go" button.
    - Click on the appeared book, it will take the user to the book's details page.
    - All the reviews that belong to the book are at the bottom.
5. As a **user**, I need to create comments for a review.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Enter the name of the book to comment on the reviews in the "Search" box and hit the "Go" button.
    - Click on the appeared book, it will take the user to the book's details page.
    - Pick a review and press the "show comments" button below it.
    - Click on "Add a comment"
    - Write your comment in the text area.
    - hit the "Submit" button
    - Navigate to the previously selected review and confirm the new comment is displayed.
6. As a **user**, I need to be able to give books a rating.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Sign in to an account, or create one
    - Enter the name of the book to comment on the reviews in the "Search" box and hit the "Go" button.

- Click on the appeared book, it will take the user to the book's details page.
- Click on the "Add a Review" button.
- Click on the "add a rating" button.
- in the drop-down below the button the "add a rating" button, select the option "overall"
- click along the row of stars.
- click the "submit" button
- find the new review, click the associated "show comments" button
- Beneath your username is a rating with the same score you entered, and the genre listed as "overall"

7. As a **user**, I need to be able to give books a rating based on a genre.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Sign in to an account, or create one
    - Enter the name of the book to comment on the reviews in the "Search" box and hit the "Go" button.
    - Click on the appeared book, it will take the user to the book's details page.
    - Click on the "Add a Review" button.
    - Click on the "add a rating" button.
    - in the drop-down below the "add a rating" button, select one of the options.
    - click along the row of stars.
    - Repeat the previous 3 steps twice more.
    - On the last rating, click the red X button to in a rating's box.
    - Confirm that the associated rating was removed.
    - click the "submit" button
    - find the new review, click the associated "show comments" button
    - Beneath your username is a rating with the same score you entered, and the genres and scores assigned during the review creation.
8. As a **user**, I need to be able to submit upvotes or downvotes on reviews.
    - Go to http://localhost:80/ on your browser, the homepage will open.

- Sign in to an account, or create one
- Enter the name of the book to vote on the reviews in the “Search” box and hit the “Go” button.
- Click on the appeared book.
- Each review has a “thumbs-up” and “thumbs-down” icon on the left. Click on the “thumbs-up” button.
- Confirm the count between the voting buttons increated
- Click on the “thumbs-down” button.
- Confirm the count between the voting buttons decreased

9. As a **professional journalist**, I need to have review highlighting.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Sign in to an account, or create one
    - Click the username at the top right of the screen
    - Select the ‘Are you a Professional Journalist?’ box
    - Enter a journalist name
    - Enter the name of a book in the “Search” box and hit the “Go” button.
    - Click on the appeared book.
    - Click the ‘Add a review’ button
    - Enter something in the text area
    - Select the ‘Highlight review box’
    - Submit the review and wait for the refresh
    - The review should be at the bottom with yellow highlighting at the top of the review
    - Click ‘show comments’
    - The top of the review should have ‘Professional Reviewer: [username]’
10. As an **author**, I need to be able to set an official description for a book.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Sign in to an account, or create one

- Click the username at the top right of the screen
- Select the ‘Are you an Author?’ box
- Enter an author name
- Enter the name of a book that has an author matching the user’s author name, in the “Search” box and hit the “Go” button.
- Click on the appeared book.
- Click the edit icon button
- Enter something in the text area
- Hit update and wait for the refresh
- The new description should appear on the book display page

11. As an **author**, I need to be able to set an official genre for a book.
    - Go to http://localhost:80/ on your browser, the homepage will open.
    - Sign in to an account, or create one
    - Click the username at the top right of the screen
    - Select the ‘Are you an Author?’ box
    - Enter an author name
    - Enter the name of a book that has an author matching the user’s author name, in the “Search” box and hit the “Go” button.
    - Click on the appeared book.
    - Click the edit icon button
    - Select a genre from the dropdown
    - Hit update and wait for the refresh
    - The new genre should appear on the book display page

2.2 Test Completeness

The testing goal will be to have at least **80% line coverage** of the **core systems**, with particularly **strong focus on key logic branches**. Tests should **run automatically before a deployment**, preventing a build being pushed with failing tests. All **test-breaking bugs** are fixed immediately such as to make the tests pass and the build deploy. All **non-testbreaking bugs** will be logged as issues and given a severity according to the disruption it could cause to the development processes or how dangerous/inconvientant it is in the production environment.

3 Resource & Environment Needs

3.1 Testing Tools

As of yet we haven't started using any advanced tools. What follows are the basic technologies that we are using to run our tests
Tools we are using:

- jUnit
- Jest

3.2 Test Environment

It mentions the minimum **hardware** requirements that will be used to test the Application.

Example, following **software's** are required in addition to client-specific software.

- Latest Ubuntu release
- Github Actions

4 Terms/Acronyms

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

TERM/ACRONYM	DEFINITION
API	Application Program Interface
DAO	Data Access Object