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

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

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	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	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: Missing everything
Input: ReviewUploadInput object missing every field
Expected output: an ArrayList with all 3 previously indicated errors

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 Id
Input: CommentUploadInput object with missing Id
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

Acceptance Tests

Regression Tests

Load Tests

2.2 Test Completeness

The testing goal will be to have at least **80% 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