TEST PLAN FOR

TEXT SAVVY

ChangeLog

Version	Change Date	By	Description
1.0.0	Feb. 26, 2022	All Members	Initial Testing Plan
2.0.0	April 2, 2022	All Members	New Workspace Tests

1 Introduction

1.1 Scope

This is our testing scope for Sprint 3:

1. Account Management
    - Create an account
    - Login to an account
2. Highlight and Save Text
    - Obtaining the correct text
    - Obtaining the source of the text
    - Manually add a text
    - Manually add a source for a text
    - Delete a text
    - View the full text in a popup
3. Browser Extension
    - Context menu
        - Access to workspaces

        - Add highlighted text to a workspace
    - Extension drop down functionality
        - Main website redirection
4. Create a Workspace
    - Add a workspace
    - Edit a workspace's name
5. Sharing a Workspace
    - Add a collaborator to a workspace
    - Delete a collaborator from a workspace
6. Test only in Chrome

1.2 Roles and Responsibilities

Name	Net ID	GitHub username	Role
Faith de Leon	deleonkf	@deleonkf	Back-end Developer, Test Manager
Reymel Eusebio	eusebior	@r3ym3l	Back-end Developer. QA Analyst
Marielle Manlulu	manlulum	@mariellemanlulu	Front-end Developer
Joshua Moreira	moreiraj	@OfficialArms	Front-end Developer, QA Analyst
Emmanuel Valete	valetee	@valetee	Front-end Developer

Role Details:

1. Front-end Developer
    - A developer focused on creating the components that make up the UI.
2. Back-end Developer
    - A developer focused on the database and the database controllers, which links the backend and the frontend.
3. QA Analyst
    - Responsible for testing the quality of the application before it gets merged to the develop branch.
4. Test Manager
    - Plans and manages the testing effort for the application.

2 Test Methodology

2.1 Test Levels

Core Feature: Account Management

Unit Tests:

1. Make sure that a GET request for an existing Auth0 ID returns a 200 response status and exactly one Account object that matches the given Auth0 ID.
2. Make sure that a GET request for an account that does not exist returns an empty response body.
3. Make sure that a POST request with a valid account returns a 200 response status and a response body with the new account object.
4. Make sure that a POST request with an invalid account returns a 400 response status.
5. Make sure that creating an account that follows the schema defined in accounts.model.js is successful and throws no errors.
6. Make sure that creating an account that does not follow the schema defined in accounts.model.js throws an error, with a message about which attribute is missing.
7. Make sure that a new account is only added to the database if the user is new (no duplicate accounts added).
8. Make sure that the correct username of the user that is logged in is displayed on the sidebar.
9. Make sure that the "Logout" button is visible in the Sidebar.
10. Make sure that the "Manage Account" button is visible in the Sidebar.
11. Make sure that a PATCH request for an existing account returns a 200 response status and updates the correct field(s).
12. Make sure that a PATCH request for an account that does not exist returns a 400 response status.
13. Make sure that a GET request for an existing email returns a 200 response status and exactly one Account object that matches the given email.
14. Make sure that a GET request for an email that does not exist returns an empty response body.

Integration Tests:

1. Make sure that Auth0's loginWithRedirect() function is called if no user is logged in.
2. Make sure that the Sidebar and Dashboard components are visible when a user is logged in.
3. Make sure that no components are visible when a user is not logged in.
4. Make sure that clicking the "Logout" button on the Sidebar calls Auth0's logout() function.

Acceptance Tests:

1. As a user, I want to be able to create an account so I can log in from any device.
    - Given I'm in the role of guest user
    - I open the Text Savvy webpage
    - Since I do not have an account, I click on the Sign Up button.
    - Then the system shows me the Sign Up form containing "Email address" and "Password" fields, which are required.
    - I fill in the "Email address" field with a valid Email address.

- And fill in the “Password” field with my password
- Then I click the “Continue” button
- Then the system should create an account for me.
- The system shows an authorization page.
- When I click on the “Accept” button, the system shows the home page of Text Savvy, indicating a successful login.

2. As a user, I want to be able to login to my account to view my saved texts and workspaces.
    - Given I’m in the role of a registered user
    - I open the Text Savvy webpage
    - The system shows the Login to Text Savvy form containing “Email address” and “Password” fields, which are required.
    - When I fill in the “Email address” field with my registered Text Savvy Email address
    - And I fill in the “Password” field with my password
    - Then click on the “Continue” button,
    - The system shows the home page of Text Savvy, indicating a successful login.

Core Feature: Chrome Extension

Unit Tests:

1. Make sure that the button in the popup renders and contains the text “Go to Text Savvy”.
2. Make sure when the button is clicked, it creates a new tab.
3. Make sure when the button is clicked, it redirect you to the web page server
4. Make sure that formatText function adds ellipses to the text on notification when the length of test is >= 20 characters.
5. Make sure that formatText function does not add ellipses to the notification when text length is < 20 characters.
6. Make sure to trim the text with a length of > 20 characters to 20 characters.
7. When a user tries to click a workspace from the context menu, ensure that it prints an error message when matching ids or workspace is > 1.
8. When a user tries to click a workspace from the context menu, it should print an error message when there is no matching id.
9. When a user tries to click a workspace from the context menu, they should not print an error message when there is one (and only one) matching id.
10. Ensure entering an empty string would still save the text.

Acceptance Tests:

1. As a user, I want to be able to highlight text from a different site and save the text into a workspace.

- Given I'm in the role of a registered user
- I navigate to a website with some text I am interested in saving later
- I highlight the interesting text and right click
- In the context menu that pops up, I hover my mouse over "Add to Workspace", then select the workspace I would like to add the text to.

Core Feature: Highlight and Save Text

Unit Tests:

1. Make sure that a GET request for all of the texts returns a 200 response status and a response body with all of the text objects.
2. Make sure that a POST request with a valid text returns a 200 response status and a response body with the new text object.
3. Make sure that a POST request with an invalid text returns a 400 response status.
4. Make sure that a DELETE request with an existing text returns a 200 response status and a response body with the string "Text Deleted."
5. Make sure that a DELETE request with a text that does not exist in the database returns a 400 response status.
6. Make sure that creating a text that follows the schema defined in texts.model.js is successful and throws no errors.
7. Make sure that creating a text that does not follow the schema defined in texts.model.js throws an error, with a message about which attribute is missing.
8. Make sure that the TextList component is visible once all of the data has been loaded.
9. When viewing the pop up of a text box, make sure that clicking outside of the pop up closes it.
10. When viewing the pop up of a text box, make sure that clicking the exit button closes the pop up.

Acceptance Tests:

1. As a user, I want to be able to trace back to the source of the text saved.
    - Given I'm in the role of a registered user
    - I open the Text Savvy webpage
    - I navigate to a text that I'm interested in
    - I can click on the chain symbol on the text box and it will copy the link to my clipboard
    - Or I can click on the text and a pop-up will appear in which I can also copy the link to my clipboard
    - Once the link is in my clipboard, I can just paste it in a chrome browser
2. As a user, I want to be able to manually add text to the database.
    - Given I'm in the role of a registered user

- I open the Text Savvy webpage and log in to my account
- I navigate to the card that says “Add Text” and click on it
- In the large box that says “Enter Text…” I type in the text that I want to save, which is required.
- In the small box below, I can add a source to the text, which is not required.
- I then click the arrow icon to add the text to the database.
- A new card should pop up with my text in it, indicating a successful add.

Core Feature: Create a Workspace

Unit Tests:

1. Make sure that a GET request for all of the workspaces returns a 200 response status and a response body with all of the workspace objects.
2. Make sure that a GET request for a workspace with the specified ID returns a 200 response status and a response body with the correct workspace object.
3. Make sure that a GET request for a workspace with an invalid ID returns a 400 response status.
4. Make sure that a POST request with a valid workspace returns a 200 response status and a response body with the new workspace object.
5. Make sure that a POST request with an invalid workspace returns a 400 response status.
6. Make sure that a PATCH request to update an existing workspace’s name returns a 200 response status and updates the name.
7. Make sure that a PATCH request for a workspace that does not exist returns a 400 response status.
8. Make sure that creating a workspace that follows the schema defined in workspaces.model.js is successful and throws no errors.
9. Make sure that creating a workspace that does not follow the schema defined in workspaces.model.js throws an error, with a message about which attribute is missing.
10. Make sure that the selected workspace’s name is shown on the Dashboard.
11. Make sure that the Sidebar contains the green header with the ‘+’ button, which allows users to add a new workspace.
12. Make sure that clicking on the ‘+’ button on the Sidebar causes the input field to be visible.
13. Make sure that pressing “Enter” when adding a new workspace in the Sidebar calls the POST function to add it to the database.
14. Make sure that a workspace’s name is fully displayed if its length is less than or equal to 20.
15. Make sure that a workspace’s name is display trimmed if its length is greater than 20.

Integration Tests:

1. Make sure that the SidebarWorkspace component is visible when a user is logged in and they have workspaces. This should show a list of the user's workspaces.
2. Make sure that the TextList component is visible when a user is logged in and they have selected a workspace. This should show a list of the workspace's texts.
3. Make sure that clicking a SidebarWorkspaceItem calls the handleOnClickWorkspace() function in the SidebarWorkspace component. This should make the selected workspace's texts visible.
4. Make sure that clicking the edit icon on a SidebarWorkspaceItem calls the handleOnWorkspaceEdit() function in the SidebarWorkspace component. This should make the workspace settings pop up visible.
5. Make sure that the AddWorkspaceIcon component is visible when a user is adding a new workspace (i.e., they clicked the '+' button).

Acceptance Tests:

1. As a user, I want to be able to create a new workspace so that I can organize my saved texts.
    - Given I'm in the role of a registered user
    - I open the Text Savvy webpage
    - I can click on the Plus symbol next to the "My Workspaces" label.
    - When I type in the name of my new workspace in the new field that says "Enter Workspace Name"
    - And press enter
    - The new workspace will appear in the sidebar.

Core Feature: Share Workspace with Others

Unit Tests:

1. Make sure that a GET request for a list of the workspaces with the given owner ID returns a 200 response status and a response body with all of the workspace objects.
2. Make sure that a GET request for a list of the workspaces with an owner ID that does not exist returns a 400 response status.
3. Make sure that a GET request for a list of the workspaces that the given email is a collaborator of returns a 200 response status and a response body with all of the workspace objects.
4. Make sure that a GET request for a list of the workspaces with a collaborator that does not exist returns a 400 response status.
5. Make sure that a PATCH request to update an existing workspace's collaborators returns a 200 response status and updates the list of collaborators.
6. Make sure that a newly added collaborator's email should have a '*' at the end if the user has not saved their changes.

7. Make sure that a newly added collaborator's email should have no '*' at the end if the user has saved their changes.
8. Make sure that the 'Save' button is visible after adding a new collaborator.
9. Make sure that an error message is shown when a user tries to add a collaborator with an invalid email.
10. Make sure that an error message is shown when a user tries to add a duplicate collaborator.

Integration Tests:

1. Make sure that clicking the trash can icon for a CollaboratorItem calls the handleRemoveCollaborator() function in the WorkspaceSettings component. This should remove the collaborator from the Workspace Setting's list of collaborators.

Acceptance Tests:

1. As an owner of a workspace, I want to be able to invite other users to my workspace.
    - Given I'm in the role of a registered user
    - I open the Text Savvy Webpage
    - I hover my mouse over the workspace that I want to add another user to,
    - And click on the pencil button that appears beside the name of the workspace.
    - In the popup that appears,
    - I type in the email address of the user that I want to add to my workspace,
    - And click on the plus icon beside the email address field.
    - Then click the save button that appears in the bottom of the popup.

Regression tests:

- All unit tests and integration tests are executed for each commit pushed to the develop and main branch.

2.2 Test Completeness

Testing will be complete when:

- 65% test coverage
- All automated and manual test cases executed successfully
- All open bugs are fixed or will be fixed in next release

3 Resource & Environment Needs

3.1 Testing Tools

General Tools/Methods:

- Github Actions (automation for tests)
- Github Issues for tracking bugs and task features
- IDE debugger (i.e. VSCode Debugger)
- Testing Library
    - Jest

Front-end:

- Chrome DevTools (Inspect Element)
- Libraries:
    - React Testing Library
    - React Test Renderer
    - Jest
    - sinon-chrome
    - Enzyme

Back-end:

- Libraries:
    - React Testing Library
    - Jest
    - Supertest

3.2 Test Environment

The minimum **hardware** requirements that will be used to test the Application:

- Machine with operating system agnostics:
    - Windows 10 or above, or
    - MacOS

The following **software's** are required in addition to client-specific software:

- Node JS v16.14.0
- Chrome Browser 93 and above

4 Terms/Acronyms

TERM/ACRONYM	DEFINITION
API	Application Program Interface
AUT	Application Under Test
QA	Quality Assurance
UI	User Interface