SE 3XA3: Test Plan
MacSidenotes

Team 4
Josh Mitchell mitchjp3
Matthew Shortt shorttmk

October 30, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Oct 30th, 2016	0.5	Sections 2, 4, 5, 6 Added

This document specifies the tools and techniques that will be used to test the adherence of MacSidenotes to its Functional and Non-Functional Requirements.

1 General Information

1.1 Purpose

1.2 Scope

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
Abbreviation1	Definition1
Abbreviation2	Definition2

Table 3: **Table of Definitions**

Term	**Definition**
Note	Text created by the user. Usually intended to be saved and viewed at a later time.
Sidebar	The popup window that appears when a user clicks on the MacSidenotes icon. In the sidebar: users write, save and delete notes, as well as view the master list of notes.
Master List	The list containing all previously saved notes. It can be viewed by clicking the List button.

1.4 Overview of Document

2 Plan

2.1 Software Description

MacSidenotes is a Chrome extension that allows users to create notes for a webpage and save them alongside that page's URL. They can then view all previous notes in a list and navigate to a URL to continue their previous work.

2.2 Test Team

The following project members will be responsible for writing and executing tests on MacSidenotes:

- Josh Mitchell
- Matthew Shortt

2.3 Automated Testing Approach

Given the UI-driven nature of this project, much of the automated testing will be focused on user interaction, which can be simulated with our Testing Tool QUnit.

2.3.1 Whitebox Testing

Values held in the program will be examined before and after a simulated click or keyboard event, to ensure proper updating of system variables. These tests are Whitebox or Structural tests, as they require knowledge of variable names and the "under-the-hood" storage architecture used by the program.

2.3.2 Blackbox Testing

The very basic functionality of MacSidenotes involves the appearance and disappearance of UI elements. Testing this functionality can be done without knowing the internal structure of the program, just the knowledge that the "Show List" button should show the user a list.

2.4 Testing Tools

QUnit will be the dominant tool for testing the JavaScript functionality of the extension. It facilitates automated unit testing, assertions, synchronous and asynchronous callbacks as well as testing user actions through simulated mouse clicks and keyboard input.

2.5 Testing Schedule

See Gantt Chart at the following links:

- .gan format
- .pdf format

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:
   Output:
   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:
   Output:
   How test will be performed:

3.1.2 Area of Testing2

...

3.2 Tests for Nonfunctional Requirements

3.2.1 Area of Testing1

Title for Test

1. test-id1

   Type:
   Initial State:
   Input/Condition:
   Output/Result:
   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:
   Output:
   How test will be performed:

3.2.2 Area of Testing2

...

4 Tests for Proof of Concept

4.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2 Area of Testing2

...

5 Comparison to Existing Implementation

The project that MacSidenotes is derived from (sidenotes) is currently unable to be tested. It relies on Dropbox's Datastore API which was retired 2 years ago, so it has been defunct since then.
That said, the basic superficial functionality of MacSidenotes can still be contrasted against sidenotes, as the original repository contains a system-level description of its abilities and a gif displaying a user's interaction with the extension.

6 Unit Testing Plan

6.1 Unit testing of internal functions

Each of the below tests requires its own driver. None require a stub.

6.1.1 deleteNote

deleteNote removes the note associated with the URL the user is currently viewing from the master list of notes.
To test: Simulate a click of the Delete Note button with QUnit, then search through Chrome's local storage to ensure no notes exist with a URL that matches the URL the "user" was viewing when deleting that note. If there is no match, the test is successful.

6.1.2 deleteEmpty

deleteEmpty removes all empty notes from the master list.
To test: Call deleteEmpty, search through local storage to ensure no empty notes exist. If no empty notes are found, the test is successful

6.1.3 clickCounter

clickCounter increments numClicks, which counts the number of times the List button has been clicked.
To test: Check the value of numClicks, simulate a click of the List button, check to see if numClicks has incremented by 1. If so, the test is successful.

6.1.4 showList

showList displays the master list of notes to the user. It also removes it from the screen if it is already present.
To test: Simulate a click of the List button, check the visibility of the List element. Simulate another click and check the visibility again. If they do not match, the test is successful.

6.1.5 updateMasterList

updateMasterList appends the note the user has saved to the master list so it can be viewed when the List button is clicked.
To test: Simulate writing a note and clicking the Save Note button. Simulate a click of the List button and check to see if an element of that list exists with the same content and associated URL of the typed note. If both the content and the URL match, the test is successful.

6.1.6 updateNote

If the user has previously saved a note for the URL they are currently viewing, updateNote will display it when the extension icon is clicked.
To test: Simulate the writing and saving of a note. Close MacSidenotes and click on the icon again. Check the value of the sidebar's text area against the typed note. If the content of the text area matches the typed note, the test is successful.

6.1.7 saveNote

saveNote saves the current note in local storage along with it's associated URL.
To test: Simulate the writing and saving of a note. Check local storage to ensure that the master list includes a note that contains the same content and associated URL as the typed note. If both the content and the URL match, the test is successful.

6.1.8 getURL

getURL returns the URL of the webpage the user is currently viewing.
To test: Call getURL and check it's value against Chrome's official code for grabbing URLs found here

6.2 Unit testing of output files

MacSidenotes does not create output files, as everything is contained within the Chrome browser's local storage.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.