SENG 426 Assignment 2

June 15, 2017

Andrew Xu - V00791274
Tyler Becker - V00169236
Brian Pattie- V00757655
Yuanxi Zeng - V00819686
Jonah Rankin - V00808910

Introduction

Automated functional tests were implemented for the ACMEPass functionality of the ACME Security System web application using Selenium and JUnit. This report describes the scenarios covered by automated functional tests, the defects discovered, and a summary of the testing metrics and the time required to fix any uncovered defects.

Functional Tests

The functional tests implemented are intended to test all functionality of the ACMEPass feature of the ACME Security System website. All implemented functional tests begin with signing into the ACME Security System website and going to the ACMEPass page.

The features and cases tested are summarized in Table 1.

Features Tested	**Test Cases**
View ACMEPass list	View the ACMEPass list: 1. Check that there is one table element 2. Check that there is one ID column
Create ACMEPass	Create an ACMEPass by and check it is visible in the ACMEPass table 1. Clear All ACMEPasses from the database 2. Create an ACMEPass 3. Read the first row of the ACMEPass table to see that the credentials match the created entry.
Create ACMEPass failure	Ensure that an ACMEPass cannot be created without providing a Site. 1. Click Create New ACME Pass button 2. Enter Login and Password but no Site 3. Assert that the Submit button is not enabled
Edit ACMEPass	Edit an ACMEPass from the ACMEPass table 1. Create an ACMEPass; 2. Find the edit button of the created ACMEPass and click; 3. Give the ACMEPass with new content; 4. Check if the content of the ACMEPass is changed accordingly in the table
Delete ACMEPass	Delete an ACMEPass from the ACMEPass table 1. Create an ACMEPass; 2. Record the size of ACMEPass table; 3. Find the delete button of the created ACMEPass and click 4. Record the size of ACMEPass table; 5. Check if the size of ACMEPass table reduces by one
Generate Password	In the Create/Edit modal test that choosing a generated password sets the password field on the Create/Edit modal

	1. Open the Create/Edit modal and click the generate password button. 2. Read the password generated 3. Press the Use button on the Generate Password modal 4. Assert the password in the Create/Edit modal matches the password used in the Generate Password modal
Generated password setting	Setting the password length on the Generate Password feature provides a password of that length. 1. Open the Generate Password modal 2. Toggle off lower case, digits, and set length as 10 3. Generate a password 4. Assert that the generated password has no lowercase characters, numbers, and 10 characters long.
Toggle password visibility	1. Toggle visibility on the ACMEPass page. On the first entry of the ACMEPass table perform: a. Check that password is hidden b. Click visibility button c. Check that password is visible d. Click visibility button e. Check that password is hidden 2. Toggle visibility on the Create or Edit Password modal. Edit the first entry in the ACMEPass table and perform: a. Check that password is hidden b. Click visibility button c. Check that password is visible d. Click visibility button e. Check that password is hidden
Sorting	1. Sort ACMEPass table by ID: a. Clear the acmepass table of the MySQL database b. Insert 2 rows of known sorting order c. Check that the ID of the first row is “1” d. Click the table header labeled “ID” e. Check that the ID of the first row is “2” f. Clear the acmepass table of the MySQL database 2. Sort ACMEPass table by website: a. Clear the acmepass table of the MySQL database b. Insert 2 rows of known sorting order c. Click the table header labeled “Site” d. Check that the site name of the first row is “allthebeans.com” e. Click the table header labeled “Site” f. Check that the site name of the first row is “zombie.com” g. Clear the acmepass table of the MySQL database

	3. Sort ACMEPass table by login name: a. Clear the acmepass table of the MySQL database b. Insert 2 rows of known sorting order c. Click the table header labeled “Login” d. Check that the login name of the first row is “alphonse” e. Click the table header labeled “Login” f. Check that the login name of the first row is “zula” g. Clear the acmepass table of the MySQL database 4. Sort ACMEPass table by password: a. Clear the acmepass table of the MySQL database b. Insert 2 rows of known sorting order c. Click the table header labeled “Password” d. Check that the ID of the first row is “2” e. Click the table header labeled “Password” f. Check that the ID of the first row is “1” g. Clear the acmepass table of the MySQL database 5. Sort ACMEPass table by date created: a. Clear the acmepass table of the MySQL database b. Insert 2 rows of known sorting order c. Click the table header labeled “Created Date” d. Check that the created date of the first row is “May 13, 2016 1:32:02 AM” e. Click the table header labeled “Created Date” f. Check that the created date of the first row is “Sep 14, 2016 10:32:02 AM” g. Clear the acmepass table of the MySQL database 6. Sort ACMEPass table by last date modified: a. Clear the acmepass table of the MySQL database b. Insert 2 rows of known sorting order c. Click the table header labeled “Last Date Modified” d. Check that the last modified date of the first row is “Oct 29, 2016 11:32:02 AM” e. Click the table header labeled “Last Date Modified” f. Check that the last modified date of the first row is “Jun 14, 2017 10:32:02 AM” g. Clear the acmepass table of the MySQL database
Pagination	1. Create enough passes(more than 20) that enables Nextpage button. a. Click Nextpage Button b. Check the number of tr bars(passes) in the next page should be the total passes minus the previous passes. c. Click Previouspage Button d. Check the number of tr bars(passes) in the previous page should be 20. 2. Create enough passes(more than 20) that enables Nextpage button.

	a. Record one ID of the row from page1 b. Click Nextpage button c. Record one ID of the 1st row from page2 d. Click previouspage button e. Record one ID of the 1st row after previouspage pagination f. Click nextpage button g. Record one ID of the 1st row after Nextpage pagination h. Check the equality of IDs after pagination

Table 1 - Summary of tested features and test cases

Before every test, the a Firefox Webdriver is loaded to facilitate interactions between Selenium and the Firefox browser that ACMEPass is opened with. The local instance of the application is opened, the login modal opened, and the login fields filled with test user Frank Paul's login credentials, "frank.paul@acme.com" for login, and "starwars" for password. Next the "Sign In" button is pressed and the ACMEPass page is loaded.

View ACMEPASS List simply checks that there is a table element, and that there is an ID column in that table.

The *Create ACMEPass* test uses the size of the table to determine how many passes are present in the table. It records the number of rows present at the start of the test by selecting all table rows and finding the size of the resulting array. Next, it creates a new pass. By clicking the "Create new ACME pass" button, filling the Site, Login, and Password fields of the dialogue, and clicking save, a new pass is added to the table. This process is used by many of the later tests described here. The new table size is recorded, and the difference is checked against the expected value of 1.

Create ACME Pass Failure tests that an ACME pass cannot be created without entering a site name. It clicks the "Create new ACME pass" button and fills the Login and Password fields, leaving the Site field blank. It then checks that the submit button is not enabled.

Edit ACME Pass starts by creating a pass as before. Next is clicks the edit button in the last column of the ACMEPass table, opening the edit modal. For each field, it clears the existing input, then enters a new site name, login, and password. The submit button is clicked, updating the database with the new values. After waiting for the table to reload, the values are scraped from the table row, and compare against the expected values, "new site", "new login", and "new password", respectively.

Delete ACME Pass performs many of the same operations as *Create ACMEPass*, but in different a order. It starts by creating a pass, then recording the number of rows in the table. The red "X" button is clicked, followed by the "Delete" button in the newly opened modal. This

should remove the pass from the table and the database. The size of the table is sampled again, and the difference checked against the expected value of “1”

Generate Password clicks the “Create new ACME pass” button, then the “Generate” button (from the pass creation form), landing in the password generation form. The default options are used, and the “Generate” button (from the password generation form) is clicked. This populates the password field of the password generation form, but not the creation form. The generated password is recorded, then the “Use” button is clicked. This should copy the password from the password generation dialogue to the ACMEPass creation dialogue. The value of the password field in the creation dialogue is checked against the recorded password to see if they match. They are expected to be equal.

Generated Password Setting follows the previous test up until the the opening of the password generation dialogue. This time, the Lower Case Letters and Digits checkboxes are clicked, unchecking them. The Length field is selected, cleared, and replaced with a value of 10. The “Generate” button is clicked, and the resulting password is checked for lower case letters and digits using regular expressions. This assertion is expected to be false. The length of the string is tested against 10, which is expected to be true.

Password Visibility Toggle Buttons exist on the ACMEPass table as well as on the Create or Edit ACME Pass modal. To test this functionality the a password is created and the input field type attribute is checked. If the type is “password” the password is hidden, when the type is not “password” the password is visible. It is asserted that passwords are initially hidden then become visible when the button is clicked and hidden again when the button is clicked a second time.

To test the sorting ability of the ACMEPass table, the testing suite clicks on each sorting button in turn and confirms that the appropriate ACMEPass is at the top. To guarantee the number of passes and their positioning, each test case clears the database before inserting 2 rows, each with a known sorting order. In each case, the rows are designed such that the only column sort that will put the desired row at the top is the column sort being tested. Every test follows the same basic flow by clearing the the database, inserting the the test data, then clicking on the appropriate sorting button, which puts it in ascending order based on that column. Sort By ID skips this click because ID is the default sort order. Next, the test checks that the top row of the table is correct by checking the field being tested (except for the Sort By Password test, which checks based on ID’s for simplicity). The sort button is clicked again to put the table in descending order, and the new top item is checked again.

Test Metrics

To analyze the efficiency of the functional tests the time to implement tests, number of defects found and the time to fix discovered defects was tracked. Test case implementation time only accounts for the time actually taken to write the tests. A significant amount of time

(approximately 6 hours) was required to configure Java IDE's and the Selenium and JUnit libraries as well as formulating a testing structure. The test design is satisfactory for a very small project which doesn't change much but is very tightly coupled to the layout of the page, a more practical solution for writing more robust tests is to use the Page Object pattern to abstract the test actions from the html elements (http://www.seleniumhq.org/docs/06_test_design_considerations.jsp).

Test Case	**Test Name**	**Time to Implement Test Case (minutes)**	**Defects Found**	**Description of Defects Discovered**	**Time to Fix (minutes)**	**Time to Run Test Case (seconds)**
View ACMEPass List	viewPassList	5	0			6.5
Toggle password visibility ACMEPass Table	tablePwdVisibilityTest	30	1	The visibility toggle has to action when clicked	30	7.5
Toggle password visibility Create/Edit Modal	editPwdVisibilityTest	15	1	The visibility toggle has to action when clicked	15	7.0
Create ACMEPass	createTest	20	0			6.5
Create ACMEPass failure	noSiteEnteredTest	10	0			6.7
Edit ACMEPass	editTest	10	0			7.6
Delete ACMEPass	deleteTest	15	1	ACMEPass is not deleted	25	7.8
Generate password	genPwdTest	15	0			6.8
Generated password setting	pwdSettingTest	20	0			6.7
Sort by ID	sortByID	15	0			7.1

Sort by Site	sortBySite	15	0			8.1
Sort by Login	sortByLogin	15	0			7.1
Sort by Password	sortByPassword	30	1	The table is sorted using the encrypted version of the password	90 (still broken)	6.4
Sort by Date Created	sortByDateCreated	15	0			7.1
Sort by Date Last Modified	sortByDateModified	15	0			6.6
Pagination		90	0			15.8

Table 2 - Test Development and Defect Fix Times

We were unable to implement a fix to the password sorting defect. This was due to time constraints and unfamiliarity with the Java Persistence Architecture and how passwords are encoded but we were able to determine the cause of the defect.

Metric	**Value**
Number of test cases	16
Number of defects found	4
Percentage of test cases that revealed defects	25%
Total test case execution time	121.3 sec
Total test case implementation time	335 min
Average test case execution time	8.7 sec
Average test case implementation time	24 min
Total time to correct/fix/pinpoint defects	160 min
Average time to correct/fix/pinpoint defects	40 min

Table 3 - Test Metrics

Conclusion

Overall, the effort required to design and implement the tests was not a very effective means of identifying system defects. Many of the defects found were discovered through exploration and smoke testing. The Selenium tests produced a large number of false positives. One of our team members experience significant inconsistencies in the success of the **editTest** test case and it would pass about 30% of the time when all tests were executed but would always pass when run alone or performed manually. Furthermore the tests are very brittle, in an environment where the application is continuously evolving, such as in Agile development, it must be ensured that there is a layer of abstraction between the test logic and the page elements, like the Page Object pattern mentioned earlier. The least obvious defect found through the functional tests was the incorrect sorting of the password field, which is sorted by the encrypted value rather than the plain text value, unfortunately we were unable to fix this defect since we could not determine the mechanism for encrypting and decrypting passwords.