# **CS 1632 – DELIVERABLE 3**

### Project Group Members:

### Bayne, Brennan M. – [bmb101@pitt.edu](mailto:bmb101@pitt.edu)

Github Username – bbayne

GitHub Repo URL : <https://github.com/bbayne/CS1632Deliverable3>

**Introduction, Issues, and Problems**

Overall, I do not feel that I faced too many issues on this project. Most of the requirements were relatively easy to test, though the biggest issue arose on the Cathedral page. One requirement is absolutely impossible to truly test: the images must be of the cathedral. Sure, one can check the ‘alt’ attribute, as that attribute is designed to display a description of the image in the event that the image doesn’t load or the user places their mouse overtop of it. But, it is a poor excuse for a true check of the image’s contents. It would also require rigid naming practice from the website’s designers – if they ever changed the way they described their images, the test would instantly break.

Over the summer, during my internship at Northrop Grumman, I used a program called SikuliX to automate the STP for a product known as the Scalable Node Architecture. SikuliX runs using computer vision, built into Python scripts. It is as simple as passing an image into a method, and SikuliX finds that image and acts upon whatever object is found accordingly. For a situation like this one, I would have used SikuliX to run an existence test, passing in a picture of the cathedral and then lowering the ‘required similarity’ variable such that, as long as the image on the page depicts an object that looks like the cathedral of learning, the test would pass.

In other words, what I am saying is that Selenium is by no means the right choice for testing a visual requirement (“the images on this page MUST reflect the cathedral and nothing else”). And, if the makers of this website continue to use it in spite of these requirements, they will run into issues with fragile tests later on.

Quite frankly, I feel that, for a black-box test such as this, SikuliX or something similar would have been far more effective in general. Graphical interface testing tools can be finicky (Jubula, for example, doesn’t bother to tell its users that it doesn’t actually see the screen, but rather just uses locations…a very dangerous way to test indeed), but a tool like Sikuli which is entirely based on searching the screen for images and interacting with the found elements accordingly is a much more effective way to test on the black-box level.

As for the rest of the project, problems were few and far between. I found most of the requirements to be relatively easy to test, though I do feel that many of them were far too vague. For example, the “Hello” page is supposed to display “Hello from <trailing\_value>” for every possible value. I created a test that checks whether or not this works for values such as “…/hello/jazz/hands”. The browser will, of course, treat this as a separate page, but the requirements suggest that ALL inputs should work for this even though, honestly, I’m not even sure that’s possible.

Some other issues with ambiguity lie just in the generality of many of the requirements. “A message should be displayed.” Well, where should it be displayed? Is it to be assumed that ONLY that message should be displayed? The home page includes more messages than requirement 1 asks for – are those messages to be accepted? Requirements 3 and 4 suggest that the values should be ‘shown to the user.’ They say nothing of how (except for an “e.g.”, which for the record is not a way of saying “This is how the message should appear”). Finally, both calculative pages suffered from serious ambiguity insofar as their valid ranges were concerned. The Fibonacci and Factorial page requirements both suggest that users should be allowed to “enter a positive integer from 1 to 100” without specifying whether or not this is inclusive. I assumed that it was, based on the start at 1, but the fact remains that I would, in the real world, have no way of knowing for sure what the customer wanted. All of this led to uncomfortable test writing in which I had to actually go to the website and check how certain output was formatted before I could even write a test for it.

In short, the above is basically just to suggest that my second biggest issue was simply that the requirements are by no means comprehensive, meaning that my tests had to be written based partially on the formatting the website already had, meaning I was testing that formatting against…nothing. It is an uncomfortable feeling, indeed, to have to assume that something is correct simply because the whole web app is doing it.

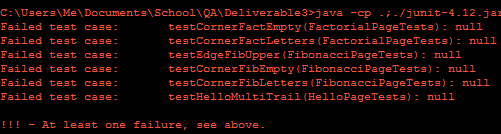
Moving on, we can talk briefly about which tests failed before we move on to the actual defect listings. 6 tests failed, those being the following:

1. Empty string input test on factorial page
2. Non-numeric (string) input test on factorial page
3. Upper edge test (input = 100) for Fibonacci page
4. Empty string input test on Fibonacci page
5. Non-numeric (string) input on Fibonacci page
6. Multiple trailing values test on Hello page

The ‘Defects’ section will go into detail for each of these failures, but the basic gist is that neither of the calculative pages could handle invalid input that was non-numeric. The Fibonacci page fails to correctly treat valid input at 100 (exploratory testing showed that the function fails for values > 30). Finally, the Hello page fails to properly parse trailing values if there is more than one (that is, more than one /<value> - so inputs like “…/hello/jazzy/jazz”).

You can observe each of these failures in the output shown below (each test case is identified by its name).

**Output**



**Defects**

SUMMARY: WebApp crashes when an empty string is submitted to the factorial page form

DESCRIPTION: Test Case ‘testCornerFactEmpty’ reports an error in which the web app crashes

when an empty string is submitted to the factorial page form (in violation of

requirement #5, which states that all invalid input of any kind should be handled by informing the user that the factorial of <given value> is 1).

REPRODUCTION STEPS:

1. Access <https://cs1632ex.herokuapp.com/fact>
2. Click the “Submit” button without entering anything into the textbox

EXPECTED BEHAVIOR:

A message is displayed reading that the factorial of is 1 - (That’s two spaces

between “of” and “is,” since we entered an empty string.).

OBSERVED BEHAVIOR:

The WebApp crashes and the only message shown reads “Internal server error"

SUMMARY: WebApp crashes when a non-numeric character is submitted to the factorial

page form

DESCRIPTION: Test Case ‘testCornerFactLetters’ reports an error in which the web app crashes

when the letter ‘a’ is submitted to the factorial page form rather than a number

(this is in violation of requirement #5, which states that all invalid input of any

kind should be handled by informing the user that the factorial of <given value>

is 1).

REPRODUCTION STEPS:

1. Access <https://cs1632ex.herokuapp.com/fact>
2. Type ‘a’ into the textbox
3. Click the “Submit” button

EXPECTED BEHAVIOR:

A message is displayed reading that the factorial of a is 1

OBSERVED BEHAVIOR:

The WebApp crashes and the only message shown reads “Internal server error"

SUMMARY: WebApp Fibonacci page treats upper-limit value as invalid.

DESCRIPTION: Test Case ‘testEdgeFibUpper’ reports an error in which the web app returns an

incorrect value for the Fibonacci of 100, the upper limit of the values which the

user may input into the Fibonacci form. (this is in violation of requirement #4, which states that users should be allowed to enter a positive integer from 1 to 100 and then be shown the fibonacci of the value). It is important to note that the web app is treating this input essentially as invalid, as it is returning a value of 1, which is what it is supposed to do for invalid inputs.

REPRODUCTION STEPS:

1. Access <https://cs1632ex.herokuapp.com/fib>
2. Type ‘100’ into the textbox
3. Click the “Submit” button

EXPECTED BEHAVIOR:

A message is displayed reading that the Fibonacci of 100 is

354224848179262000000

OBSERVED BEHAVIOR:

A message is displayed reading that the Fibonacci of 100 is 1

SUMMARY: WebApp crashes when an empty string is submitted to the Fibonacci page form

DESCRIPTION: Test Case ‘testCornerFibEmpty’ reports an error in which the web app crashes

when an empty string is submitted to the Fibonacci page form (in violation of

requirement #5, which states that all invalid input of any kind should be handled by informing the user that the fibonacci of <given value> is 1).

REPRODUCTION STEPS:

1. Access <https://cs1632ex.herokuapp.com/fib>
2. Click the “Submit” button without entering anything into the textbox

EXPECTED BEHAVIOR:

A message is displayed reading that the fibonacci of is 1 - (That’s two spaces

between “of” and “is,” since we entered an empty string.).

OBSERVED BEHAVIOR:

The WebApp crashes and the only message shown reads “Internal server error"

SUMMARY: WebApp crashes when a non-numeric character is submitted to the Fibonacci

page form

DESCRIPTION: Test Case ‘testCornerFibLetters’ reports an error in which the web app crashes

when the letter ‘a’ is submitted to the Fibonacci page form rather than a number (this is in violation of requirement #5, which states that all invalid input of any kind should be handled by informing the user that the factorial of <given value> is 1).

REPRODUCTION STEPS:

1. Access <https://cs1632ex.herokuapp.com/fib>
2. Type ‘a’ into the textbox
3. Click the “Submit” button

EXPECTED BEHAVIOR:

A message is displayed reading that the Fibonacci of a is 1

OBSERVED BEHAVIOR:

The WebApp crashes and the only message shown reads “Internal server error"

SUMMARY: WebApp crashes when the “Hello” page is accessed with a trailing value that

contains the ‘/’ character

DESCRIPTION: Test Case ‘testHelloMultiTrail’ reports an error in which the web app crashes

when more than one ‘value’ is appended to the ‘hello’ page URL. According to requirement 7, if a trailing value is provided to the URL, then we should see a message reflecting this. This is meant to work for ALL INPUT VALUES. That includes slashes, which are typically treated as new pages. When more than one word separated by slashes is submitted as a trailing value, the web app crashes.

REPRODUCTION STEPS:

1. Access <https://cs1632ex.herokuapp.com/hello/jazz/hands>

EXPECTED BEHAVIOR:

A message is displayed reading “Hello CS1632, from jazz/hands!”

OBSERVED BEHAVIOR:

The WebApp crashes and the only message shown reads “Page not found"