The UROP project I am working on is a web application test automation project. GenomeSpace is the name of the website the program is testing on. It is a cloud based storage which act like a bridge to connect the Nectar Research Cloud with the Genomics Virtual Laboratories. A GenomeSpace user can mount the swift container from his/her Nectar cloud and then send the files in the container to the virtual laboratories for processing. GenomeSpace also provide basic file manipulation functionalities like move, copy, rename, delete, etc.

The test automation program is basically to test if the functionalities are working as intended and generate report of the test outcome.

The functionalities that are tested include:

* user registration
* user login
* mount a swift container
* disconnect a swift container
* import a file using the expired public URL
* generate public URL of a file and access the file with the URL
* file rename
* copy file between directories
* copy file between containers
* move file between directories
* move file between containers
* delete a file
* lunch a GVL instance with files
* generate a DOI of a file for publishing

The program is written in Python and the libraries used in this program to aid the testing are Selenium and Unittest.

Selenium provide functionalities of simulating the website browsing process, like open up a browser, go to a page, type something in the text fields and click a link to go to the linked page. Initially, the user interface was tested with Selenium by conducting a series of events, waiting for the response and asserting the outcome was as expected. In order to do so, elements of the web page needed to be located for the events to be conducted on. However, most of the elements in the page were dynamically generated and injected by Ajax. They didn’t have any id, any name or anything that can be uniquely identified. Some of them are even merely with tag names. It was almost impossible to locate the element without hardcoding some numbers and the hardcoded code are always fragile and easy to break. That was why most of the test cases are not about the interface now.

As the functionalities of GenomeSpace are service oriented. The communications between clients and servers are the most crucial part. Instead of testing the user interface, the tests, except for registration and log-in, has changed to be sending requests and checking if the response is correct.

At the setting up stage of the program, Chrome driver, which was installed in advance, or Firefox driver, which came along with Selenium package are started and the a window shows up, then Genomespace website (<https://genomespace.genome.edu.au/jsui>) is opened. The program looks for the cookie stored under the same directory as itself is in and restore it, if there is any, for skipping the log in test and speed up the debugging process of the program itself, as the web drivers would always start a new window with no cookies, no cache and the login test is relatively slow. The program is then ready to test.

The biggest problem encountered when setting up was that the web drivers would always start a new window with no cookies and no cache. Logging in was needed every time running the program to pass the authentication and this takes a relatively long time, which slowed down process of debugging the program itself.