Project Assignment 3+4 Report and Documentation

By

Dylan J. Santa Cruz

DJS528

Software Testing and Quality Assurance

Dr. Tanmay Bhowmik

April 2, 2020

Project Assignment 3+4 Report and Documentation

This project report and documentation was completed to address the criteria and guidelines placed by the professor for the assignments given. The information within is broken down into sections addressing the questions and information requested by the assignment's rubric.

Required Information and Specifications:

Name:	Dylan J. Santa Cruz
NetID:	DJS528
GitHub Username:	Dylan-SC
GitHub Repository Link:	https://github.com/Dylan-SC/swqa-project3

Project Report Outline:

Deployment Pipeline Discussion:

Steps Associated with Pipeline (OVERVIEW):

Active-Branch

The code is developed using **Sublime 3 Text Editor**

Once changes have been saved, **Github Desktop** is used

Github Desktop stages changes as a source control manager

Once uploaded, then **Travis CI** is used to run tests

Travis CI is responsible for the Unit and End-to-End testing

The results of these tests are then uploaded to **Coveralls**

Sider is used to run static analysis tests on open pull requests

Once testing is successful, **Heroku** automatically deploys Staging app

Master Branch

GitHub merges changes from the active branch to Master branch

Travis CI runs tests on the master branch ensuring it is stable'

The results of these tests are documented by **Coveralls**

Once testing is successful, **Heroku** deploys the stable product

This deployment is what clients would use

Deployment Pipeline Discussion (continued):

Tool Information and Specifications:

Sublime 3:

Website: https://www.sublimetext.com/

Usage: Text Editor

GitHub Desktop:

Website: https://desktop.github.com/

Usage: Source Control Application

Travis CI:

Website: https://travis-ci.org/

Usage: Continuous Integration

Coveralls:

Website: https://coveralls.io/

Usage: Code Coverage Reporting

Sider:

Website: https://sider.review/dashboard

Usage: Static Code Analysis

Heroku:

Website: https://dashboard.heroku.com/apps

Usage: Cloud Hosting Platform

Benefits of the Tools Utilized:

Sublime 3 Text Editor –

Offers the ability to create and edit multiple different file types. The program offers syntax highlighting so that the information being worked is more easily legible and understandable by the user. Although the editor does not have a compiler by default, this functionality was not necessary for what it had been used for.

Github Desktop –

This is an application that simplifies the development workflow for its users. The application provides an interface which users can navigate in order to manage their repositories. For the scope of this project, it was used to commit and stage changes for the code developed. It interfaces directly with GitHub hosting services, which is fundamental for the pipeline process used for this project.

Travis CI –

This is the service used to provide the project with continuous integration capabilities. It allows the ability to constantly merge small code changes at a frequent setting. It allows for a more streamlined testing process by allowing smaller changes to be tested consistently, thus improving the workflow of the project and ensures a healthier development cycle. It has the capability to interface directly with GitHub services to provide a more seamless user experience.

Coveralls –

This utility is a means of providing a user with code coverage analytics. This coverage measures the extent to which the source code has been executed and tested during the testing process. This interfaces with Travis CI which provides Coveralls with detailed statistics and results from the code tested. The major benefits of using this product is the ability to integrate itself with Travis CI while simultaneously offering an aesthetic approach to the user.

Benefits of the Tools Utilized (continued):

Sider –

This tool is used for the static code analysis functionality for the pipeline process. Slider is used to automatically review the source code and offers insight to the code's integrity. It will check style violations, the code's overall quality, securities and dependencies. This information is provided as a comment on the GitHub pull request. The ability to interface with GitHub is what also makes this tool a beneficial choice for the project.

Heroku –

This platform as a service offers the capability to deploy changes so that they can be delivered as a service. Heroku specializes in apps, meaning that their focus is to make an app developer's experience as seamless as possible. It allows developers to "build, deliver, monitor, and scale apps". This is the final step in the pipeline process which allows the developed and tested code to be utilized by the end-user.

Tool Kit Challenges Encountered:

General Challenges:

The primary challenge faced during the completion of this project was the knowledge required in order to interface all the tools with one another. Finding these tools, which could address the requirements placed by the project, was a large issue in and of itself. However, the actual difficulty was ensuring that the tools could coordinate with one another and integrate so that the pipeline process could work efficiently and without intervention.

Once the tools were decided upon, the process of connecting the utilities led to some issues. The first of which was the language in which the application was developed in. In order to receive universal acceptance from the applications used, the source code must be supported by each process in the pipeline. This meant that I would need to recreate the entire assignment 2 so that the code can be processed more easily.

I had initially developed the assignment 2 using C++ but had to change this to Python. This allowed for better integration. Once developed in python, an external compiler could be used which converts the python into JavaScript, so that the rest of the process can use JavaScript to complete the pipeline instead of the original python. This was so that the webservice can be more easily created, as python is not a web development language, whereas JavaScript is.

GitHub Desktop and Travis CI Challenges:

The next issue was interfacing the tools themselves. Starting with GitHub, the desktop application was downloaded onto my windows system, as it was easier to use this than a CLI on my Ubuntu laptop. Once the repository was created and the files were pushed, then Travis CL could be attached to the pipeline process. In order to have this utility working, you must create a file in the repository that tells Travis CL what it needs to do. As stated previously, the difficulty was in the learning curve. It was a bit confusing to determine how to setup the Travis CL file so that it can interface with the repository properly. The file needs to have certain dictation to determine what tests are to be ran and how they are to be ran, as well as the dependencies needed in order to run the tests initially.

Toolkit Challenges Encountered (continued):

Coveralls Challenges:

Coveralls presented me with another issue. That is, that it was difficult to properly interface the results between the tests committed by Travis CI to Coveralls so that they could be properly documented. Upon fixing the naming conventions of some of my files and updating the syntax to accommodate for coveralls being added, I was able to link coveralls to my GitHub repository. It was not working at first because Travis CI was not building properly. Once it was, Coveralls was able to run correctly.

Sider Challenges:

Surprisingly enough, the static code analyzer that I used, Sider, was not an issue to get running. I connected the repository, it scanned through the code, and it would identify simple errors and mistakes. It would even tell me when there was spelling mistakes in my comments. That was nice, thankfully it was not difficult to implement either, I just needed to include a file in the root folder of the repository so that it knew what settings it needed to run with.

Setup and Execution Instructions

GitHub – Create an account and download the GitHub desktop application. Create a repository. Create, commit, push files within the repository as necessary. Have two separate branches. The master branch, which is for the latest stable build, then an active branch for ongoing development.

Travis CI – Interface Travis CI into the pipeline by logging into the dashboard with your GitHub account. Select the repository that you would like to use Travis CI for. Once added, you need to add the (.travis.yml) file to your repository. You alter the file to incorporate different methods of testing. The syntax within the file determines the language to test and run, as well as the dependencies required to run the tests. It also launches the correct corresponding test files so that the tool can run properly.

Coveralls – Similar to how Travis CI is implemented. Once Travis CI is working and building correctly, you can log into the Coveralls website and link your repository. Once the repository is linked and Travis CI is building correctly. As long as you have a reference in the Travis CI file to report the data to Coveralls, then the Coveralls dashboard should autoupdate each time your repository is updated, and tests are ran.

Sider – This toolkit also uses your GitHub login information. Once on the website, login with your GitHub account information and select the repository that you would like Sider to analyze. Within the repository root directory, you need to add another file similar to the one added for Travis CI. This file will tell Sider how and what it needs to be looking for throughout the repository. You create the file as a (.sider.yml) and then customize and tune the code within as necessary.

Heroku – This requires that you log in using your GitHub account. Once logged in, you are brought to the dashboard. You can create an app, which is similar to creating a repository in GitHub. Once you create an app, you can have two stages for each application. I used the two branches in my repository for the two stages in the Heroku app. The staging stage is where the active branch should be deployed, as it is testing the functionality of what is currently being developed. The production stage is the stable build, so it should be your master branch. At a certain point in development, the master branch should always be stable.

Cloud Platform Usage

Implementing Heroku

Heroku was one of the most difficult tools to get running correctly. I decided to initialize the toolkit and create an app called “project3-piping”. There were two stages of the application. The first one was called STAGING, and the other one was called “PRODUCTION”.

The STAGING stage was pulling from my active branch in GitHub, so it would take the branch currently being edited and try to launch an application if it passed all the tests. Otherwise, it would throw an error and not build and deploy. Once working correctly, and if the app deploys, that means the active branch can be merged into the master branch.

The PRODUCTION branch is the stable build for the application. It should be the most up to date stable code produced by the active branch. This is what app clients would be using, whereas the STAGING branch is what developers are using for testing.

Challenges Overcome

The difficulty of this part of the project came with ensuring the cloud platform was using the correct version of each component that I used within my project. At first, Heroku was building correctly, but it was not deploying correctly.

The initial problem was that there was not a buildkit specified. I changed the settings to use a python build kit. Since the bulk of my project was using python as a web development resource. Upon doing this, some of the errors were relieved and the build process was functioning. The next issue arose when the project was building but not deploying. This was due to the built application not knowing where to pull from initially. I would need to specify the correct homepage file. Once the correct file was interfaced with Heroku, the web application began to build and deploy correctly.

Testing and Test Cases Discussions:

Automated Unit Testing:

The unit tests should be called upon by the Travis CI process within the pipeline. It will use predefined function inputs which will be tested to make sure that only the correct data can be entered and that the correct output is outputted.

For the BMI calculator, there are tests to get the height and weight of the individual. During this, different tests determine that only the correct information type within a reasonable range gives the correct output.

For the Retirement calculator, there are tests to get the salary, savings goal, and age of the individual. During this, different tests determine that only the correct information type within a reasonable range gives the correct output.

Automated End-to-End Testing:

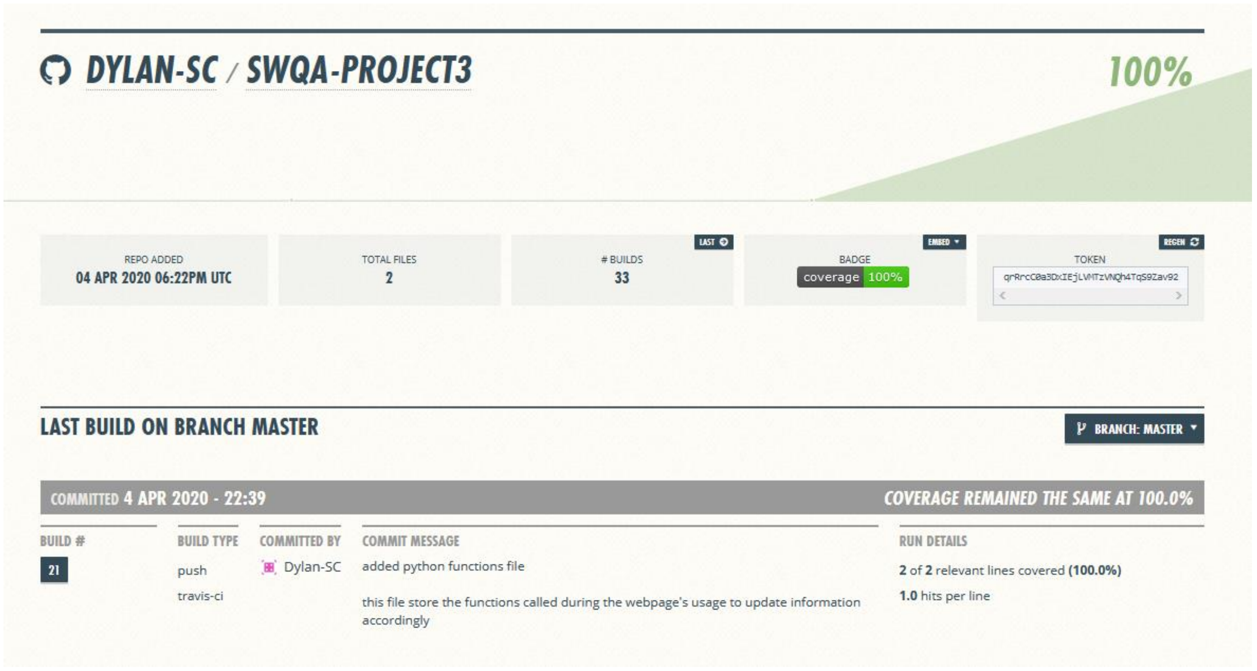
Within this section of the testing, which is to be conducted during the Travis CI step in the pipeline. The webpage functionality will be tested automatically using functions created by the developer. These test cases will run through the webpage and execute commands with specific input values, as to ensure the correct corresponding action is taken.

These functions will check that all the elements are present that need to be in the webpage. They will also check to make sure that the actions of each element are working properly by testing both and invalid and valid input for each element.

Test Cases for Manual Testing:

The test cases for the manual tests will be conducted similarly to the automatic unit tests, except that they will be conducted by the developer. Each input field will be tested within both invalid and valid inputs to determine whether or not the correct response is given.

Code Coverage Report



DYLAN-SC / SWQA-PROJECT3

100%

REPO ADDED	TOTAL FILES	# BUILDS	BADGE	TOKEN
04 APR 2020 06:22PM UTC	2	33	coverage 100%	qrRrcC0a3DxIEjLVMTzVNQh4TqS9Zav92

LAST BUILD ON BRANCH MASTER

COMMITTED 4 APR 2020 - 22:39 — COVERAGE REMAINED THE SAME AT 100.0%

BUILD #	BUILD TYPE	COMMITTED BY	COMMIT MESSAGE	RUN DETAILS
21	push travis-ci	Dylan-SC	added python functions file this file store the functions called during the webpage's usage to update information accordingly	**2** of **2** relevant lines covered **(100.0%)** **1.0** hits per line