Dr Businge, Project Coordinator
Garrett Prentice, Project Manager

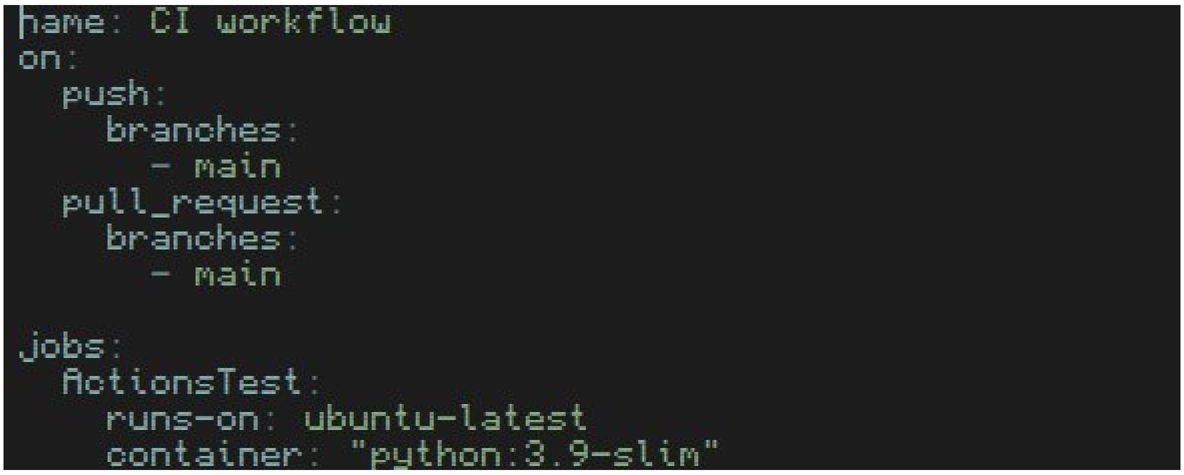
RE: CI/CD GitHub Actions

This report contains the initial setup of Continuous Integration (CI) with GitHub Actions, an explanation of the work-flow components, and how this improves the efficiency of the red/green/refactor development cycle. The repository can be found at this link: (https://github.com/S1robe/CS472-tdd)

Continuous Integration

GitHub actions require the creation of a project directory: ".github/workflows/workflow.yml". The most basic work-flow looks like figure 1 below. It contains no steps and will not pass for any commit because it does nothing. Every work-flow requires a name, and uses events (push, pull_request) in order to respond and perform this particular action. In figure 1, "push" and "pull_request" are both "events". The last section of this work-flow is the "jobs" tab. The "jobs" tab is a way to group actions to a particular platform. This is where the actual actions will be. The "runs-on" attribute specifies the type of operating system or environment that the actions are performed in. The "container" section is the type of sub environment that the program will be run on.

Figure 1: Basic Workflow.yml

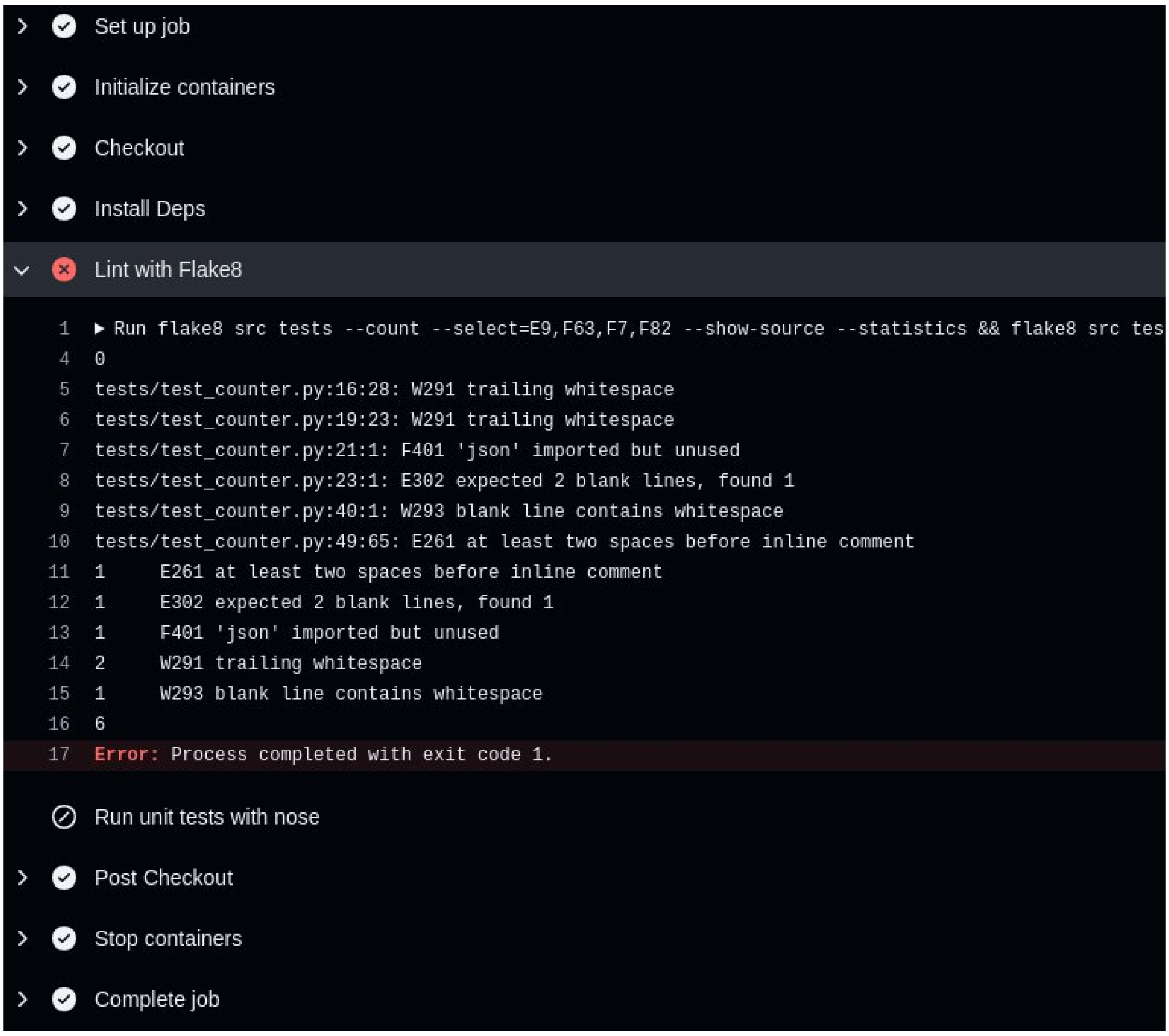


```
name: CI workflow
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  ActionsTest:
    runs-on: ubuntu-latest
    container: "python:3.9-slim"
```

After adding the steps to the workflow, GitHub actions will be able to run the code provided. As shown below, GitHub is now able to run the code from the project repository. But, like shown in figure 2, the code is not up to the standards set by 'flake8'. Flake8 is a linter which provides code quality and formatting coverage.

Figure 2: GitHub Actions Fail Lint



```
▶ Run flake8 src tests --count --select=E9,F63,F7,F82 --show-source --statistics && flake8 src tes
0
tests/test_counter.py:16:28: W291 trailing whitespace
tests/test_counter.py:19:23: W291 trailing whitespace
tests/test_counter.py:21:1: F401 'json' imported but unused
tests/test_counter.py:23:1: E302 expected 2 blank lines, found 1
tests/test_counter.py:40:1: W293 blank line contains whitespace
tests/test_counter.py:49:65: E261 at least two spaces before inline comment
1     E261 at least two spaces before inline comment
1     E302 expected 2 blank lines, found 1
1     F401 'json' imported but unused
2     W291 trailing whitespace
1     W293 blank line contains whitespace
6
Error: Process completed with exit code 1.
```

Like mentioned before, in Figure 2, the code in the repository is not up to standard with the ‘flake8’ program. Because of this, the code must be reformatted until this action passes before the next task can begin. Fortunately, the code provided does pass the tests. What is shown above only refers to formatting of the code.

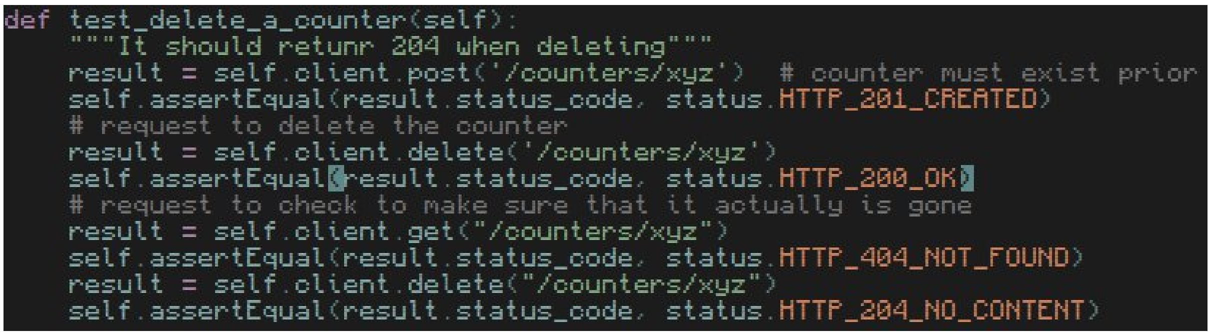
Red-Green-Refactor – Task 3

This section contains the process of development for adding the tests to delete a counter from within a python flask application. In this section there are subsections that detail the 3 phases of development, red – tests fail; green – tests pass; refactor – remove duplicate code, write efficient code, etc.

Red Phase #1

The first red phase is encountered when adding the following test cases. This is to create a response to delete an existing counter from the Flask web app. This is done in accordance with the REST API, which specifies that code is well documented and covers all possible cases. Figure 3 is the test cases written for this implementation.

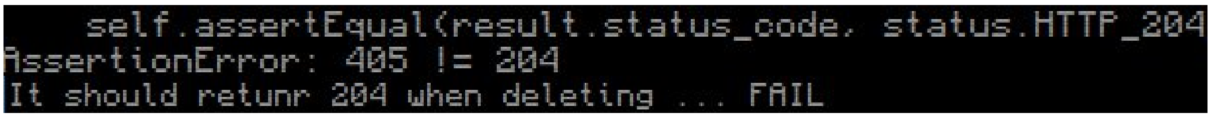
Figure 3: Test Cases #1



```
def test_delete_a_counter(self):
    """It should retunr 204 when deleting"""
    result = self.client.post('/counters/xyz')  # counter must exist prior
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    # request to delete the counter
    result = self.client.delete('/counters/xyz')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    # request to check to make sure that it actually is gone
    result = self.client.get("/counters/xyz")
    self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
    result = self.client.delete("/counters/xyz")
    self.assertEqual(result.status_code, status.HTTP_204_NO_CONTENT)
```

This test case, as shown in Figure 3, covers the requirements mentioned before. A counter must exist before it can be deleted, if it exists, then we can delete it, and we ensure that it is gone by trying to query it which should return a not found. Then we ensure that delete can handle being called on a non-existent counter, which should return "NO_CONTENT".

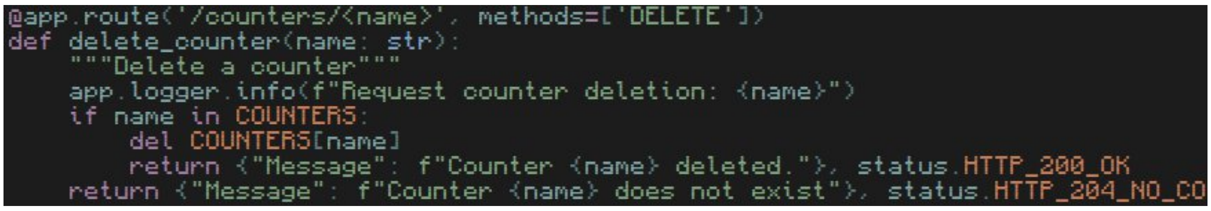
Figure 4: Red Phase #1



```
    self.assertEqual(result.status_code, status.HTTP_204
AssertionError: 405 != 204
It should retunr 204 when deleting ... FAIL
```

Like shown above in figure 4, the test cases fail, so development must continue until the test case passes in entirety. In this case, the test case is expecting 204, but received a 405, indicating that the delete was not allowed.

Figure 5: counter.py #1

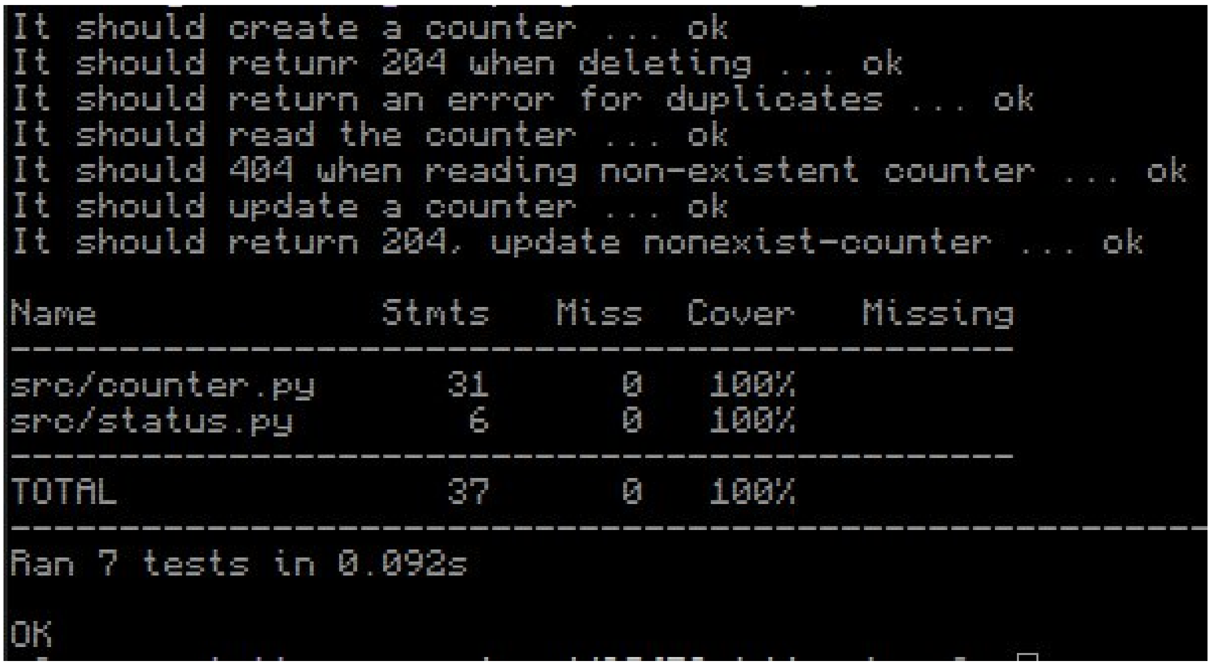


```
@app.route('/counters/<name>', methods=['DELETE'])
def delete_counter(name: str):
    """Delete a counter"""
    app.logger.info(f"Request counter deletion: {name}")
    if name in COUNTERS:
        del COUNTERS[name]
        return {"Message": f"Counter {name} deleted."}, status.HTTP_200_OK
    return {"Message": f"Counter {name} does not exist"}, status.HTTP_204_NO_CO
```

Based on the requirements that the test cases require the code above in figure 5, is an implementation that fulfills them. If a counter exists it will be deleted, if one does not exist then a 204 "NO_CONTENT" error is returned.

Green Phase #1

Figure 6: Green Phase #1



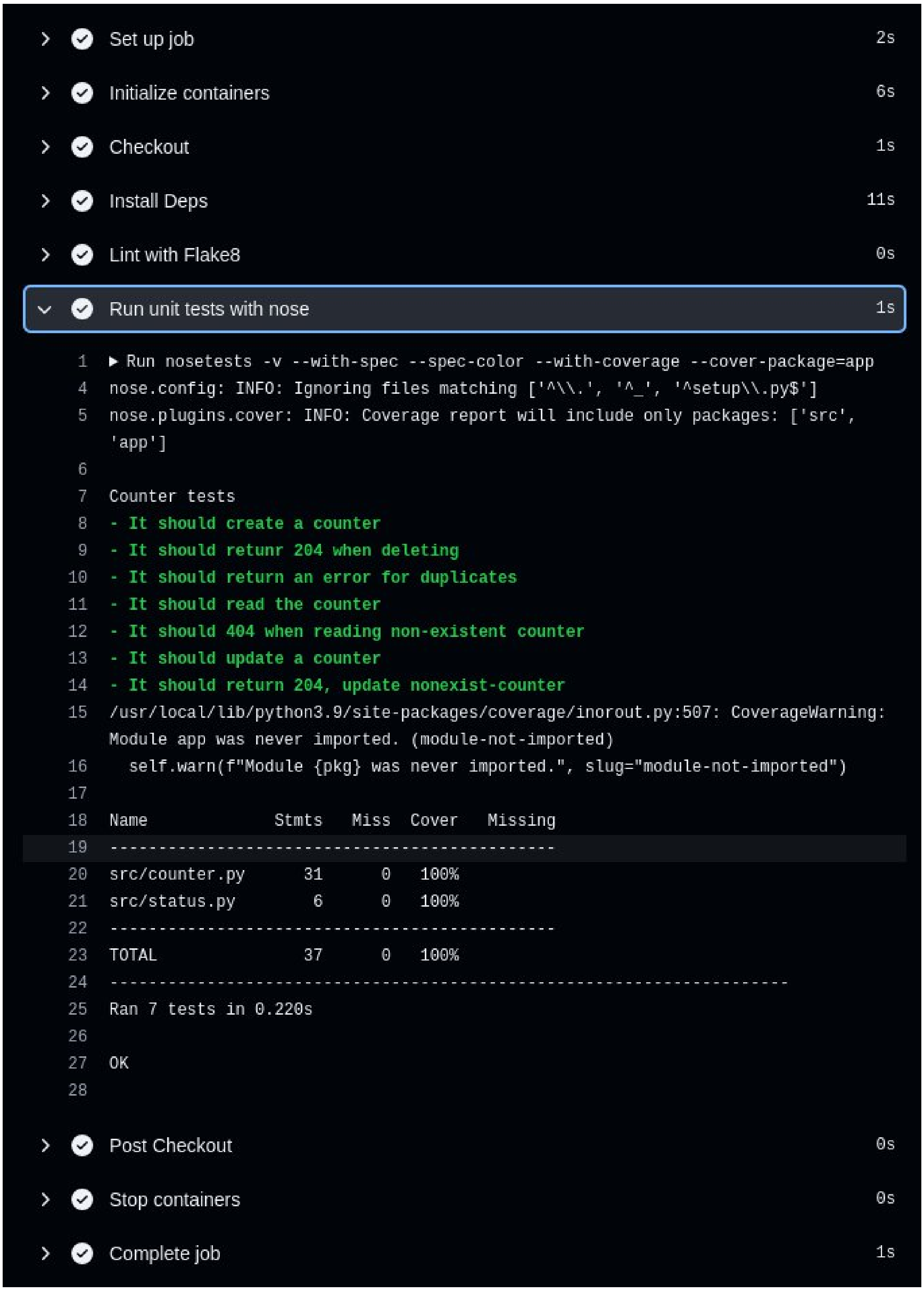
```
It should create a counter ... ok
It should retunr 204 when deleting ... ok
It should return an error for duplicates ... ok
It should read the counter ... ok
It should 404 when reading non-existent counter ... ok
It should update a counter ... ok
It should return 204, update nonexist-counter ... ok

Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
src/counter.py          31      0   100%
src/status.py            6      0   100%
--------------------------------------------------
TOTAL                   37      0   100%
------------------------------------------------------------
Ran 7 tests in 0.092s

OK
```

As shown above, in figure 6, all tests cases pass with 100% coverage. Because all test cases pass, the code before in figure 5 sufficiently fulfills the requirements established by the test cases. Furthermore, as a demonstration of completeness, figure 7 is provided on the next page. All test cases pass in the CI pipeline. This is a powerful tool for autonomously testing commits or pull requests made to a code base.

Figure 7: CI Pipeline GitHub Actions



```
Set up job                  2s
Initialize containers       6s
Checkout                    1s
Install Deps               11s
Lint with Flake8            0s
Run unit tests with nose    1s
```

```
▶ Run nosetests -v --with-spec --spec-color --with-coverage --cover-package=app
nose.config: INFO: Ignoring files matching ['^\\.', '^_', '^setup\\.py$']
nose.plugins.cover: INFO: Coverage report will include only packages: ['src',
'app']

Counter tests
- It should create a counter
- It should retunr 204 when deleting
- It should return an error for duplicates
- It should read the counter
- It should 404 when reading non-existent counter
- It should update a counter
- It should return 204, update nonexist-counter
/usr/local/lib/python3.9/site-packages/coverage/inorout.py:507: CoverageWarning:
Module app was never imported. (module-not-imported)
  self.warn(f"Module {pkg} was never imported.", slug="module-not-imported")

Name              Stmts   Miss  Cover   Missing
----------------------------------------------
src/counter.py       31      0   100%
src/status.py         6      0   100%
----------------------------------------------
TOTAL                37      0   100%
----------------------------------------------------------------------
Ran 7 tests in 0.220s

OK

```

```
Post Checkout               0s
Stop containers             0s
Complete job                1s
```