COSC 310 Software Engineering

MILESTONE 4
Testing Update Report

Due April 1, 2024

gitGoblins (L01)

Eric Harrison
Ian Steyn
Joshua Ward
Chase Winslow

Links:
GitHub Repo
GitHub Dashboard

Table of Contents

At this point in the project's life cycle, we have tested, revisited, and revised our source code multiple times, and have implemented tools to streamline our testing. In this document, you will find the following:

Testing Update

Unit Testing

Unit testing checks that functions and other code structures work as expected. All unit testing stuff is contained in the `/tests/` folder of our project. Generally, we try to stick by the following guidelines:

- One class/module ⇒ one test file.
- One function/action ⇒ one unit test
- Aim for full coverage; give tests data that covers all decision branches (though this is not always feasible)

Frameworks

Python: pytest

For Python (i.e. the bulk of our project), we are using the pytest framework because it has good documentation and is supported by Flask.

- **Configuration** is contained in `conftest.py`, which creates a temporary database, app, and other fixtures that will be used across many tests.
- **Unit Tests** are contained in files that start with `test_`.

JavaScript: Jest

For JavaScript, we are using the Jest framework, which has to be supported by npm (Node Package Manager) to work. Setting these tests up and finding tutorials for them was tricky, because we are not actually building a Node.js application. But we did it!

- **Configuration** is contained in `/weatherApp/package.json` (outside of `/tests/`)
- **Unit Tests** are contained in files with the `.test.js` extension

HTML/CSS/Jinja: our eyes/pytest

We have no framework for testing that web pages have rendered correctly, so we mostly do that by running the app and looking at the web pages ourselves. However, when we do run into aspects of web pages that need to be tested, we use pytest. For example:

- **URL routing**: do URLs route to the correct page? This is tested in `test_views.py`, which tests both the view functions and their respective HTML pages by checking whether an expected HTML component shows up for a given URL
- **Jinja Macros:** Since these contain significant logic and can accept parameters, there are some basic tests in `test_jinja_macros` that check whether expected HTML components show up based on what parameters were passed.

Running the tests

From the terminal

You have to be in the `/Code/` folder.

- `pytest` runs all Python tests
- `pytest tests/test_specific.py` runs a specific Python test file
- `npm test` runs all JavaScript tests
- `npm test specific.test.js` runs a specific JS test file

In VS Code

Some team members have also chosen to configure their VS Code “Testing” tab, which provides a nice user interface to quickly run specific tests.

Test Coverage

We are using the succinctly named tool [coverage.py] to check our Python code coverage. It supports pytest, and is very simple to use. We currently don’t have a way to see Javascript code coverage, because we have so little JS code.

Seeing Code Coverage

Before any of the options below, the command `coverage run -m pytest` must be run in the terminal.

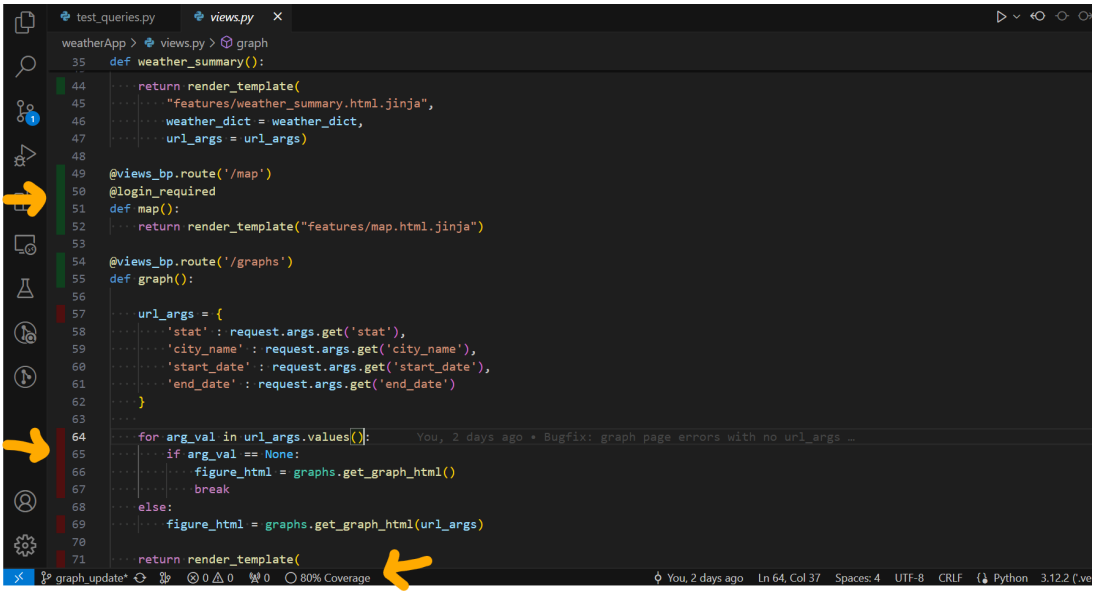
Terminal Options

- Running `coverage report` prints a short text summary in the console, containing the coverage percentage of each file, and the numbers of lines that aren’t covered
- Running `coverage html` generates an interactive report that can be opened in a browser. We don’t really use this but it is helpful to have in case the Coverage Gutters extension breaks.
-

```
C:\...\Code> coverage report
Name                      Stmts   Miss Branch BrPart  Cover
-----------------------------------------------------------
weatherApp\__init__.py       23      0      2      0   100%
weatherApp\auth.py           75      0     32      1    99%
weatherApp\db.py             68      0     24      0   100%
weatherApp\graphs.py        113     48     22      0    54%
weatherApp\queries.py        39      0     12      1    98%
weatherApp\views.py          40      8     32      0    83%
weatherApp\weather.py        20      0     10      0   100%
-----------------------------------------------------------
TOTAL                       378     56    134      2    85%
```

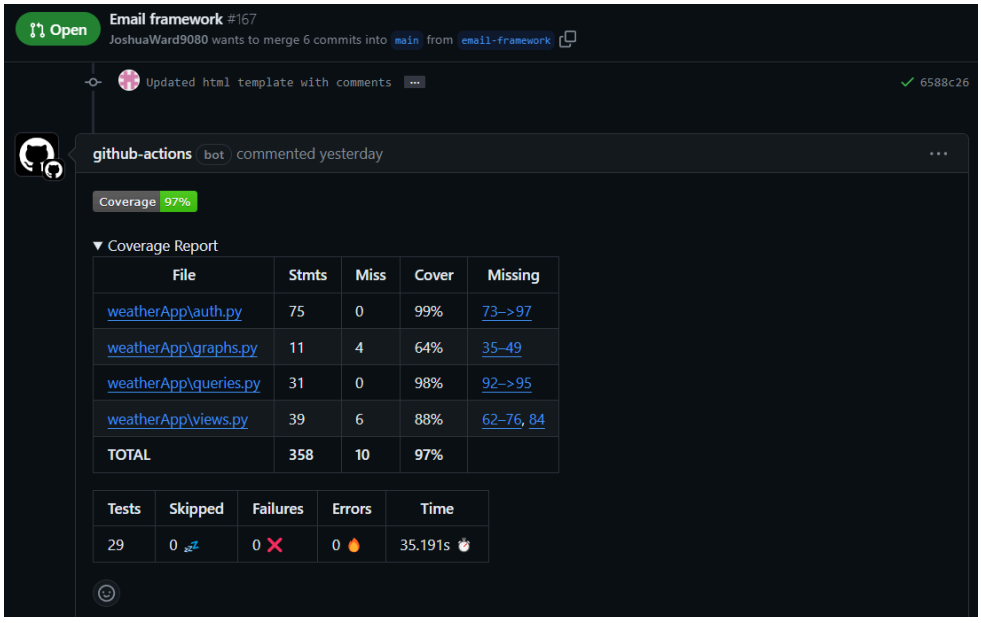
In Editor: Coverage Gutters

This is the main tool we use for visualizing coverage: the VS Code extension Coverage Gutters. After running `coverage xml`, you can open the tested files directly to see coloured highlights that indicate which lines of code are (or aren't) covered by the unit tests, as well as the file's coverage percentage.



Pull Request Coverage Comment (⚙️ Automated)

We created a GitHub Actions workflow called "Coverage Report" that uses the pytest-coverage-comment action to automatically generate a basic coverage report and comment it on new pull requests. This is useful for the reviewer, and for documenting coverage information in the remote repository. The configuration code is in `workflows/cov.yaml`.



Integration Testing (⚙️ Automated)

Our project now has two continuous integration (CI) workflows set up, which run through GitHub Actions. These workflows run automatically on every push and pull request. They install dependencies, build the program, and run unit tests on several virtual machines, so that we can test the program on different operating systems and language versions.

Essentially, these workflows catch little errors that we might not if we simply ran tests on our own machines, and it ensures that new code does not break old tests.

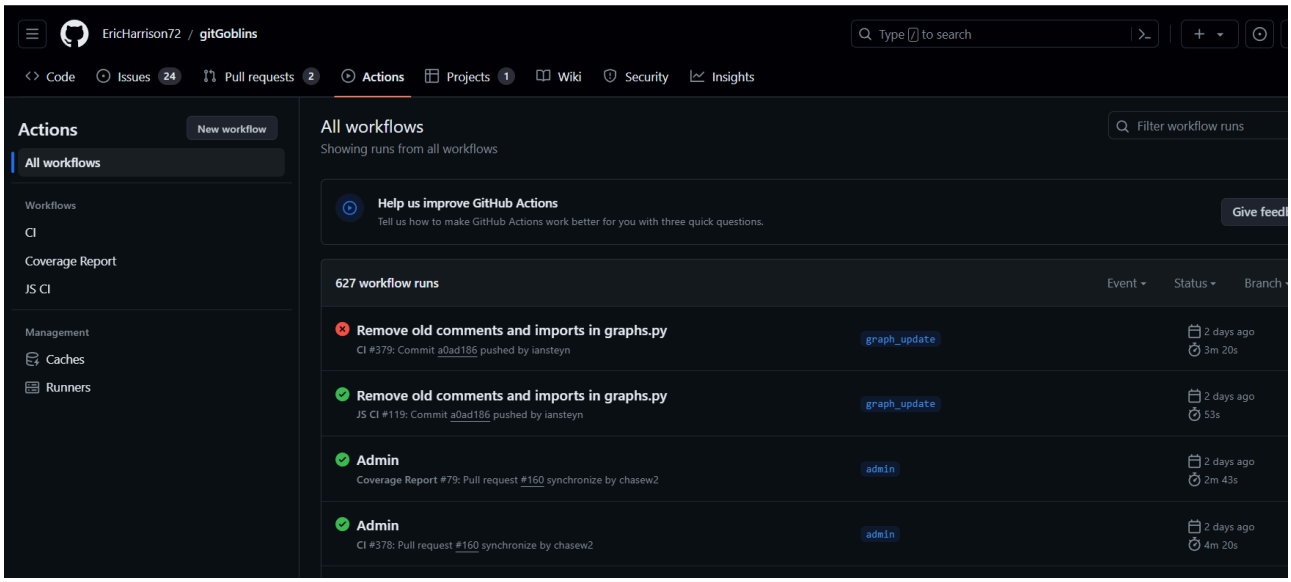
Workflows

Python: “CI”

This workflow runs all Python unit tests on the latest releases of Ubuntu, Windows, and MacOS, with Python 3.11 and 3.12. The configuration code is in `workflows/ci.yaml`.

JavaScript: “JS CI”

This workflow runs all JS unit tests on the latest releases of Ubuntu, Windows, and MacOS. The configuration code is in `workflows/jsci.yaml`.



Other Comments on Testing

What has been tested so far?

At the time of writing, the main branch has 97% coverage (at least for Python). In branches not merged to main, we also currently have incomplete tests for incomplete features: the graph feature, the email framework, and the admin role. However, nearly everything that has been implemented has been thoroughly tested, including:

- app initialization
- database setup and query functions
- Jinja macros
- URL routing and views
- user authentication
- the map feature
- general helper functions

Future Improvements

JS code coverage

As mentioned previously, we have not investigated checking test coverage for our JavaScript stuff, because we have so little JavaScript to test. However, if this project were to extend beyond the next few weeks, we would certainly do more work on this.

Edge Cases

With our final week of implementing features ahead, we will be working hard to ensure that they are all thoroughly tested. This will include checking more edge and error cases than we currently are, so we can be sure there are no app-crashing mistakes that can be made by the user.

Internal Documentation

We have a file called `testing_guide.md` where we add notes and instructions on all testing tools. We are learning so many new things at once that it is helpful to have a quick reference point when we forget how to do stuff.

General Update

The Product

Overview

The deliverable product of this project is a prototype web application that allows you to interact with Australian weather data from 2008-2017. Significant progress has been made; almost all functionalities have been implemented and tested. Though there are some requirements left to implement, the project's velocity has increased drastically in the last few weeks, and extraneous features have been cut from the backlog.

Current State

We have completed many of the core components of the application, including user authentication, location selection, the map feature, most of the graph feature, and a rain prediction model. In our final two weeks, we will be focusing on cleaning up current functionality, testing to fix bugs, preparing the project's final presentations, and implementing final requirements: the notification system and the admin role.

The Process

Teamwork

The process that the team is working in, an Agile Scrum framework, is working well to increase the visibility of each individual's work to the team, so that everyone is informed of what is going on in the project and what changes are being made. The two weekly meetings (Discord on Tuesdays and labs on Thursday) allow for collaboration, feedback, and assistance to thrive. The weekly sprints have helped keep the team on schedule by providing a constant flow of tasks and deadlines, so that everyone is always working towards the project completion.

Quality Assurance

The team has worked very hard to ensure the quality of the project's code, and has implemented strategies to do so: code reviews, testing, and refactoring.

Code Reviews

Every pull request is reviewed by at least one other team member (two if they are large), and constructive feedback is given, detailing what works, what doesn't and what might need to be changed. Reviews have been important in ensuring that our code is clean, understandable, and consistent within the project, and that each change is beneficial to the project.

Testing

Automated testing has been integrated into the GitHub repository so that each change is constantly measured against our tests over multiple operating systems, which allows the team to spot problems early before adding potentially problematic code to the main branch. See more in the Testing Update above.

Refactoring

The team regularly refactors the codebase, to clean up any code smells and ensure maintainability. The reason this happens so often is that we are constantly learning more about how to follow good software engineering practices, and we don't always get it right on the first try.

Further Improvements

Our process continually gets refined as we learn new things. One specific improvement we could make would be to create a system for logging which reviews people have done, so that we can ensure everyone has done enough for the current sprint.

[END OF REPORT]