Milestone 4 Report

Engicoders
H. Bird, B. Karacelik, J. Peters, J. Ropotar, A. Rybka
Department of Computer Science, University of British Columbia
COSC310: Software Engineering
Dr. Gema Rodriguez-Perez
April 1st, 2024

Testing Implementation

To test our project, our group chose to white box test for each code implementation and black box test any implementations to the web app pages. This project has two forms of white box testing: unit testing and integration testing. Whenever any group member makes any functionality, it has to be accompanied by a test class with unit tests for all possible inputs. This ensures that each function works as the creator intends it to. Along with unit testing everything we write, we have a continuous integration function that runs every test we have written before a pull request can be pulled into our main branch. This form of integration testing ensures that the new code works itself but doesn't break previously written code. As for black box testing, this has been recently introduced. We haven't formally utilized black box testing, but whenever new functions have been added to the webpage that users can interact with, the reviewers of the pull request are tasked to test different inputs as if they were customers and report back to the pull request. Through using the three aforementioned testing techniques, we have been able to fully test all of our code when it's written, before it's implemented, and when it's merged.

In the upcoming weeks, we plan to further black box test our project. This will hopefully prove there are no bugs in our code and if there are, find them and fix them. Through testing all corners of the project, we will be able to make a more robust system. With all the implemented tests so far, we believe our code is well-tested.

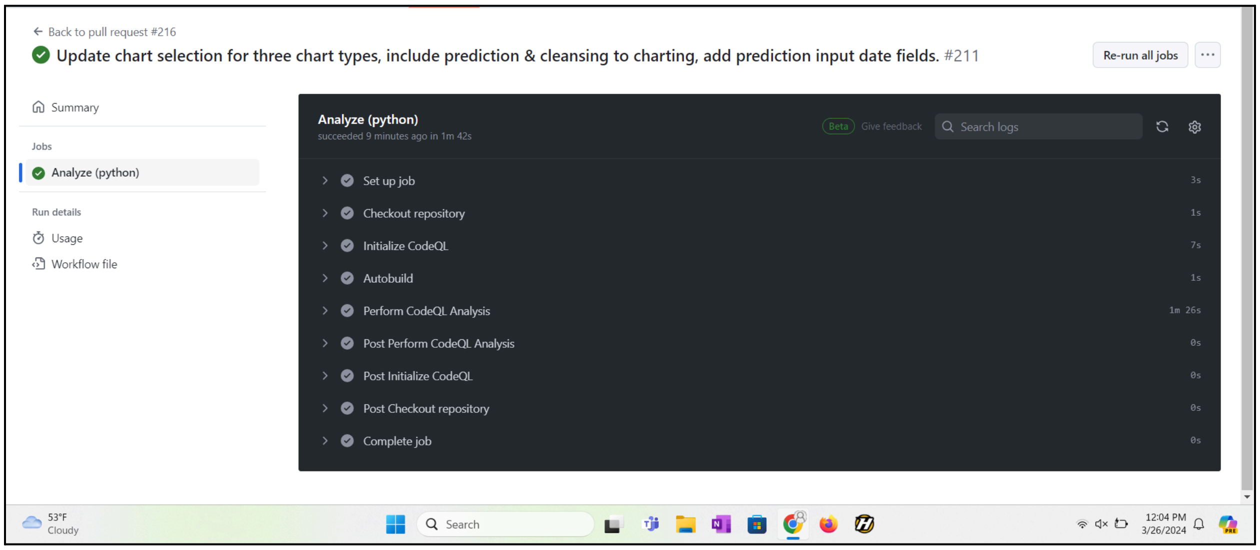
Current Functionality

The project as it stands now is nearly complete. A user accessing the site first lands on a welcome page, explaining the program and requesting the user to sign up. From there, they can log into an existing account, or register a new account. Once logged in, they can use the software with existing public ThingSpeak channels, or, if they upgrade their account to a premium user through accessing a different page with prompts, upload their data to analyze. The dataset chosen is cleaned and presented with a nice visual graph of the user's choosing. All of these features have been tested, from a requirements point of view with Pytest, as well as a user point of view. As a user, the program flows smoothly with no noticeable flaws or bugs.

Automation

Our group has implemented two forms of automation into this project. All automation is implemented using a continuous integration pipeline. This pipeline has two major features.

Feature 1: This is our CodeQL GitHub workflow. CodeQL is a powerful workflow developed by GitHub that scans for security vulnerabilities inside source code without executing this code. Because this is a front-end application this is critical to protect against SQL injection attacks. This automation feature is run every time a pull request or more merge requests is made to the main branch. The results of this check can be seen by everybody both on the PR and on the repository. The feature runs inside a virtual machine hosted on the GitHub servers. An example of a successful pull request that passed this stage of the pipeline is shown below:



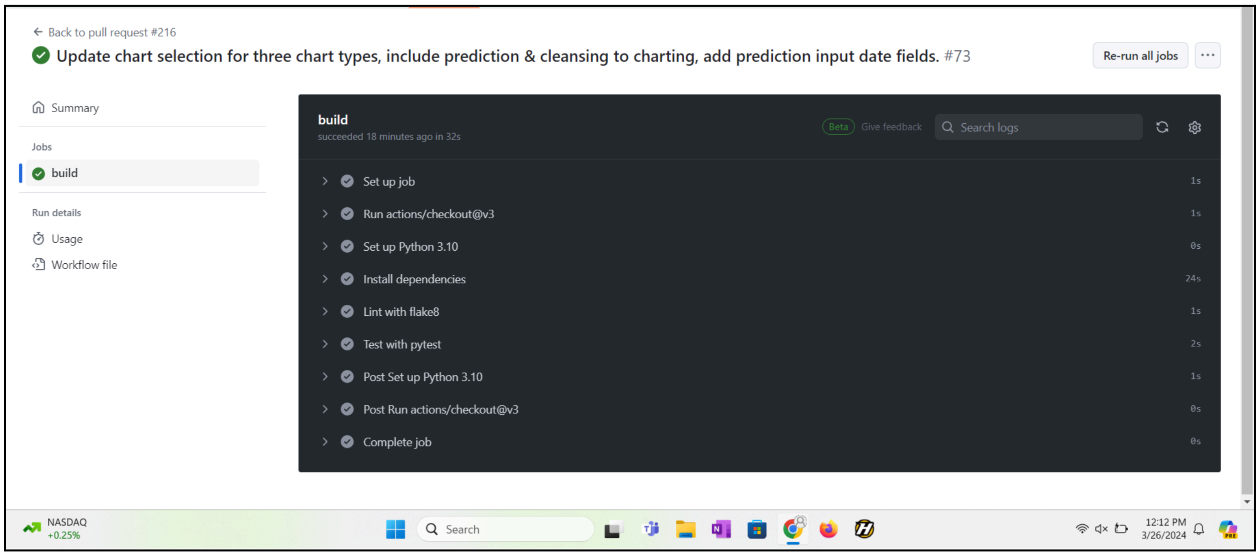
← Back to pull request #216

Update chart selection for three chart types, include prediction & cleansing to charting, add prediction input date fields. #211

```
Analyze (python)
succeeded 9 minutes ago in 1m 42s

Set up job                      3s
Checkout repository             1s
Initialize CodeQL               7s
Autobuild                       1s
Perform CodeQL Analysis     1m 26s
Post Perform CodeQL Analysis    0s
Post Initialize CodeQL          0s
Post Checkout repository        0s
Complete job                    0s
```

Feature 2: This is our testing & linting workflow. Linting is the statically analyzing code for syntax errors, bugs, and luscious constructs. Within this workflow, we use flake8 as our linting library. This workflow also tests all of our unit tests using pytest. The pytest command is run on the virtual machine which is running ubuntu-linux. This workflow is run every time a pull or merge request is made to the main branch. Using this feature the software team can verify that code is not only being tested but that the tests are passing before merging to main. This stops code that would otherwise break the build from passing review. This can be seen as a type of integration testing. An example of a successful PR that has passed the testing & linting workflow can be seen here:



← Back to pull request #216

Update chart selection for three chart types, include prediction & cleansing to charting, add prediction input date fields. #73

```
build
succeeded 18 minutes ago in 32s

Set up job                      1s
Run actions/checkout@v3         1s
Set up Python 3.10              0s
Install dependencies           24s
Lint with flake8                1s
Test with pytest                2s
Post Set up Python 3.10         1s
Post Run actions/checkout@v3    0s
Complete job                    0s
```

Finally, if either of these workflows detects issues and fails the PR they are attached to cannot be merged. This protects not only the function of our codebase but also acts to improve the quality and security of the code we write.

Project Summary

The Process:

The process that we have selected as a group, has allowed us to make improvements to the source code, without causing too many merging conflicts, and overall errors. Throughout the project process flow, we broke up the work into individual parts, so that duplication in code did not occur. Working on separate parts, allowed for more efficient workflow and bottlenecks between program files. This also allowed for less time spent rebasing and blockers to other parts of the software, while issues and pull requests were worked on. Automated testing was implemented to ensure code comprehension, and functioning Flask application. One of the group members worked on the front-end integration and formatting. Three members of the group worked on the back-end integration, which included user authorizing, home page outputs, collecting data and sequencing the data through filtration, chart configurations, routing, and alarm sensing. Finally, one member was a “free-agent” who helped on both front-end and back-end issues where needed. Each week, project goals and expectations were discussed to plan out the sprints. These sprints allowed for a clear understanding of each member's contributions and any issues that were met throughout the week. Along with the weekly sprints, a 1-2 hour collaboration time was set each week to work together to quickly problem solve potential blockers in the process. This collaboration time was implemented about half way through the project and has saved a lot of troubleshooting time. Although this process has not been perfect with a few issues blocking others and some miscommunication which ended in rebasing code, our group has become great at working together and communicating to avoid any conflicts.

Quality Retention:

As a team, we have designed a thorough quality assurance process to ensure the reliability and functionality of our source code. Through a combination of white box testing, including unit testing and integration testing, and black box testing for web app pages, we have evaluated each code implementation. The requirement for every functionality to be accompanied by a test class with unit tests for all possible inputs demonstrates the commitment to verifying the intended behaviour of each function. Additionally, our continuous integration function runs all tests before a pull request can be merged into the main branch, ensuring that new code functions correctly and does not disrupt existing functionality. Although black box testing is a recent addition, reviewers are tasked with testing new webpage functions from a user's perspective. Looking ahead, there are plans to expand black box testing to uncover any potential bugs and further improve code quality and robustness.

Additionally, our group has created a quality assurance process using automation and continuous integration to ensure the quality and security of the source code. Through the CodeQL GitHub workflow, we conduct automated scans for security vulnerabilities within the source code, crucial for protecting against SQL injection attacks in the front-end application. This workflow is triggered for every pull or merge request to the main branch, with results visible to all stakeholders. Additionally, the testing & linting workflow analyzes the code for syntax errors, bugs, and problematic constructs using flake8 as the linting library, and runs unit tests using pytest. This workflow runs for every pull or merge request, ensuring that code is thoroughly tested before merging to the main branch, preventing potentially disruptive code from passing review. The integration of these workflows not only enhances the function and security of the codebase but also reinforces the commitment to delivering a quality end product.