Get Big Get Huge

CI/CD Pipeline – Phase 1 Status Report

In this part of the CI/CD pipeline design process, we have explored various types of workflows which could facilitate good coding practices and make our work easier through basic automation. As such, we have looked into five different ideas to implement in this pipeline: linting and code style enforcement, code quality via tool, code quality via human review, unit tests via automation, and documentation generation via automation. At this point in time, we have not implemented every one of these ideas in our pipeline; we will go into why we have or have not added each one in more detail below.

First, we looked at linting and code style enforcement. There are two components to this; first, we use JavaScript Standard Style, via `standard --fix`, to enforce good code style locally. Next, in the actual pipeline, we use ESLint through GitHub's Super Linter action workflow to check each pull request to main and each push, ensuring all JavaScript and markdown files are properly styled. For this action, we have edited a few of the style choices to fit our preferences; notable mentions include Stroustrup style curly braces and not including spaces before parentheses and code blocks. By having two layers of code style enforcement, we ensure that the style of our code is well-maintained on both the local and repository levels. To us, this enforcement is important because we value consistency in our code style; when working in a team, if we do not enforce strict style rules, then one file could end up looking extremely different from another. While this does not make any functional difference, it does cause a loss of cohesion for anyone reading or maintaining the code, which is unprofessional.

Next, we considered implementing code quality via tool. Although this is important, as code quality is undoubtedly integral to well-engineered software, we did not find a suitable tool

for our needs. The recommended software was not free, which is not ideal for the purposes of our project. We will look into contacting the developers of these tools and mentioning our student status to see if they can offer us free educational versions of their tools; however, this means that our code quality review will currently be done manually.

To enforce code quality by human review, we set up branch protection for our main branch. No one can push to the main branch directly; all development will happen on separate feature branches for each feature being developed, and when these features are “finished”, we will submit a merge request. Here, at least one person besides the creator of the pull request must approve it. We chose to make this requirement only one person for now because we do not want to have progress blocked if too many people forget to check the pull requests or some other issue arises, but this may change if we decide we should have more reviewers. This review will make sure we are checking each other for developing good code, and can help increase our sense of team unity.

While we plan to implement unit testing via Jest soon, we thought that this lab was going to be on unit testing. Hence, we wanted to wait until the lab so we were more familiar with it. The lab was changed, however, so we will look into adding Jest as soon as possible.

Finally, when a branch is merged with main, we generate JSDoc and put the documentation on a Github Wiki page via Github Actions. Documentation is important, both for the sake of maintaining code and so that users have an easier time using your product. We are satisfied with maintaining JSDoc for only the main branch, as feature branches would be changing too frequently for documentation to be meaningful. By having this process automated, this encourages us to write detailed JSDoc and edit it if necessary, as we would not have to manually import our new documentation whenever we want to add or change any.