|  |
| --- |
| Hutton Smith (hds177) – GitHub: hutdude |
| Assignment 3 |
| Repo: https://github.com/hutdude/BmiCalculator |

|  |
| --- |
|  |

**Report**

My deployment pipeline begins with source control. As instructed, I am using GitHub to manage my code source. This allows me to seamlessly integrate source control into my workspace in VS code. The source control hub in VS code intuitively provides push, pull, compare, staging, and any other Git functionality I would need. GitHub also allows me to compare previous versions of my code, should I need to re-examine old code. While GitHub is a great tool, its control flow can be confusing at times. The difference between processes like forking, merging, pushing, pulling etc. can be difficult for a beginner to learn and creates a higher barrier to entry. For continuous integration, I utilized GitHub Actions, an insanely seamless and intuitive method provided by GitHub. They provide different ‘workflows’ you can add to your project repository. I added the ‘NodeJS with Webpack’ configuration which, should my website be deployed to a server, would automatically compile my npm packages into a build and upload it to the server when the repository is updated. Additionally, they allow you to customize the configuration files (YAML in my case) before uploading for additional customization. The downside to using this service over a comparable tool, such as CircleCI, is that it does not have the customizability of its peers. For example, CircleCI can be integrated with email or other workplace messaging apps to update developers on its work. GitHub Actions does not offer such functionality. For my case, though, GitHub actions works well. For static code analysis, I opted for ESLint. ESLint is easily integrated into a node app using npm commands. Then, to run the tests, all I have to do is type in a command in the terminal and it shows its results after checking hundreds of rules. The disadvantage of this is quite obviously the fact that it must be run in the command line. This isn’t too large of an issue due to the fact that this should be conducted before deployment, however, it would be more convenient if it could be implemented into GitHub as well. **Need automated unit testing here**

For automated end to end testing, I combined Cypress with a custom GitHub Actions YAML config file. The Cypress testing suite is a great way to conduct tests as it comes with a GUI that visualizes the tests I code. To set up cypress, I had to install it using npm and then configure tests in an App.cy.tsx file. From there, I can run the tests in the command line, but to automate it, I created a GitHub actions workflow that runs the tests every time new code is pushed. The advantages of such is that the tests are all very customizable and so is the running of those tests. The disadvantage is that the setup can be a bit convoluted and the cypress module can be resource intensive.