CI/CD Phase 1 Report

Our team has decided on, for the time being, adopting three pipeline tools for the CI/CD process of our project. Following brief testing to assure proper integration of those tools into the github-actions workflow, as is currently active, we have found that nothing was out of order and that all the tools are functioning at their most appropriate capacity given the present condition of our codebase. There still lingers a concern of whether some or all of the tools currently integrated will cease to work in response to changes on the configurations ruling over the current pipeline. Not to mention that the possibility of something breaking upon the addition of more tools (which could very much happen in a near future) can not exactly be ruled out as well. In light of those well-founded concerns, we have taken to establish a countermeasure expressed through the structure of our pipeline that will be able to prevent the most common of calamities from ever happening or at the very least to isolate each red flags so that they may be rectified at the moment’s notice.

Our pipeline structure is by no means a unique one, and we never really tried to make it one, to begin with. It is really simple and easy to follow, in proportion to the number of packages that are and will be included. We defined a .yaml file, required for github action to work its magic, inside of which there can be found jobs triggered by either push or pr on the one specific branch called testing. We might need to expand it so that more than one branch can activate the triggers, though given that most of the branches are still barren there is no real need to do so just yet. In the file we separated the three tools each into its own job, guaranteeing the parallel execution of the jobs; in effect, it will make sure that each error can be traced to one specific job corresponding to one specific tool. That said, restructuring of the jobs might on the table should we ever require different tools to run in sequence, but for the moment all of the tools served solely as a warning machine, and that there is no automatic corrective measure happening in the background.

Regarding the package manager, we have chosen to use yarn to manage all of our dependencies. There is no real practical reason as to why we chose yarn over npm, both uses the same package.json and the only discrepancy is the way they handle their locked dependencies which is not a real concern anyway (as a note, we did commit the yarn.lock file to the repo). The only custom yarn script defined is for jest execution, eslint runs directly from the yaml file, and jsdocs delegate the responsibility to external action. The only changes worth considering might to rid of one of the initial yarn install present in both jest and eslint jobs, but we have no idea how cross-jobs dependencies work or how safe they are considering the parallelism, so we will stash that for now.

Currently, the only independent config file in the repo is the one for eslint, It is unlikely that we will have many independent config files for additional packages, but to avoid cluttering we might have to put them in one folder. Though moving configs to anywhere but the top-level might invoke a serious path problem, so further caution is advised before making such a decision.

All in all the pipeline is working nicely and in such shape that we can scale it whenever we wish to whatever extent we desire.