Continuous Integration

Team 7: Bermuda Digital Entertainment

Prajwal Binnamangala
Joseph Sisson
Sebastian Sobczyk
Aymeric Goransson
David Ademola
CJ Donoghue

a) Summarise your continuous integration method(s) and approach(es), explaining why these are appropriate for the project. (5 marks, ≤ 1 page)

When developing our continuous integration approach for the project there were four key points that were considered. The first being the Integration machine. This was highly important because it would serve as the foundation for our CI build. We considered Circle CI and Jenkins, However we realised that the simplest way to integrate CI with our code was to use Github Actions, Which is an extension to Github that allows pre-developed CI builds to be edited to preference and inserted into project folder. This was appropriate for the project because most of the code was written on Github and therefore did not need to be tampered with as Github Actions allowed for seamless implementation of CI.

The next point considered was the Version Control Software for the CI. This was the software that would track the code and changes made and make sure that the CI build runs on the latest version of the code. There were many VSC's within Github Actions that we considered, but in the end we decided to use a combination of Super-Linter and Java CI with Gradle, the first of which is a VSC software compatible with various languages like java, Javascript, python and so on. We also made use of Java CI with Gradle because Gradle was already heavily utilised with the current code and was used to make the unit tests so we thought it would be appropriate to use Gradle as well to implement CI.

This brings us to Testing, which was the third point we considered, we wanted to make it so that every time changes were made and new code was integrated into code base, that unit tests were run automatically, This was why Java CI with gradle was also utilised in the first place, It allowed us to call the tests from the gradle test build so the test were run when new code was pushed.

The final point being automation of CI, This was the easiest target to achieve as most of the CI software available on Github Actions was automated. Meaning as long as new code was pushed, no other human intervention was needed.

b) Give a brief report on the actual continuous integration infrastructure you have set up for your project. (5 marks, ≤ 1 page)

The CI integration infrastructure we set up can be split into three parts.
As mentioned previously, we made use of Github Actions when setting up the CI for the project as it was more appropriate for a smaller scale project like ours and was easy to implement as it was on the same app as the code, Github.

The first part of the CI Infrastructure was the Super-Linter, a powerful CI tool that is compatible with various programming languages, allows code to be checked thoroughly for errors and then be integrated into the main code/code base. It is a linter (which is a code analysis tool used to flag programming errors, bugs and so on) which allowed us to check for errors even to the smallest detail. We also made use of Super-Linter as because it is so strict, it allows for better clarity, conciseness and readability of code, as the smallest errors could halt the integration process.

Then, we had the Gradle build CI which could be split into two parts, the first part was the Gradle code checker, which essentially checked the code again for errors and then integrated it into the code base. The difference between this and the super-linter being that this was more specific to Java with Gradle and less strict when it came to code syntax and grammar.

The second part of the Gradle build CI was the part that actually ran the tests, This would be run during the CI process, which begins when code is pushed/committed on Github, however, it is a separate instruction that actually calls the pre-written unit tests. It still for errors and then if there are none, the unit tests are ran and once they are all passed or no more unit tests have failed since before the code was committed, they can then be integrated into code base.