In this week's first lab session, we will continue developing our continuous integration by continuous testing; this will mean that automated tests are run whenever an integration is pushed to the main branch of the GitHub repository.

Automated tests are tests that are defined by the developer of an application which can be programmatically executed. These tests can be found in the test folder of the project from last week.

Text

Description automatically generated

So first, let's update the .circleci/config.yml to implement continuous testing in our continuous integration pipeline; this will mean our tests will run every time an integration is performed if any of these tests fail, the build will fail, telling us that our application is not working as expected.

Continuous testing is invaluable to a DevOps pipeline as it will allow us to see what features are broken/bugs that may exist with the current version of the build; this can be used to inform us that we should not release the current version of the build.

To add continuous testing, add lines 28 through 30 after the npm install command.

(Note: Software developers will write the software tests, and if tests are broken, these should be reported back to software developers, DevOps will be concerned with if exists tests to run, the command to run tests will typically be given by software developers).

Text

Description automatically generated

Before pushing to our branch, lets ensure the tests work by running "npm run test"

Once we see that our tests are working, run "git add ." to add the updated CircleCI file to our repository.

Run the command "git commit -m"added tests to continuous integration pipeline" " commit our tests added to our integration pipeline.

Run the "git push" command to update our GitHub repository.

Graphical user interface, application

Description automatically generated

We will then see our new version of the build be run and successfully executing. This tells us now our code is being merged successfully, there are no issues with the environment, and now there are no issues with the functionality of the application.

Graphical user interface, application

Description automatically generated with medium confidence

We can now see our application has a "test-run" task; if we click into this, we can see the details about the tests we performed.

Text

Description automatically generated

**Note what our build would look like if it failed due to tests**

Graphical user interface, text, application, email

Description automatically generated

If our build failed because of our tests, we would see the following build fail message; we could click on this to see the tests that were an issue if this was the case. As DevOps engineers what we will be concerned with now would be contacting the developer who made the breaking changes and getting these changes rectified.

Text

Description automatically generated

Code to replicate the broken tests in test\_utils.js:

var assert = require('assert');

var utils = require("../modules/utils")

describe('utils', function() {

describe('#add()', function() {

it('should return 2 when 1 and 1 is added', function() {

assert.equal(utils.add(1,1), 2);

});

it('should return 37 when 15 and 22 is added', function() {

assert.equal(utils.add(15,22), 37);

});

});

describe('#multiply()', function() {

it('should return 4 when 2 and 2 is multiplied', function() {

assert.equal(utils.multiply(2,2), 4);

});

it('should return 49 when 7 and 7 is multiplied', function() {

assert.equal(utils.multiply(7,7), 14);

});

});

describe('#subtract()', function() {

it('should return -4 when 4 and 8 is subtracted', function() {

assert.equal(utils.subtract(4,8), -4);

});

it('should return 12 when 4 and -8 is subtracted', function() {

assert.equal(utils.subtract(4,-8), 12);

});

it('should return 1 when 2 and 1 is subtracted', function() {

assert.equal(utils.subtract(2,1), 1);

});

});

describe('#divide()', function() {

it('should return 4 when 8 and 2 is divided', function() {

assert.equal(utils.divide(8,2), 4);

});

it('should return 1 when 1 and 1 is divided', function() {

assert.equal(utils.divide(1,1), 1);

});

it('should return 0 when 1 and 0 is divided', function() {

assert.equal(utils.divide(1,0), Infinity);

});

});

});