Purwadhika
Digital Technology School

Full Stack Web Development

Intro to Test Driven Development

Job Connector Program

Testing

Purwadhika
Digital Technology School

Testing is the process of ensuring a program receives the correct input and generates the correct output and intended side-effects. We define these correct inputs, outputs, and side-effects with *specifications*.

We have two choices when it comes to testing: manual testing and automated testing.

Testing

Purwadhika
Digital Technology School

Manual Testing

Manual testing is the process of checking your application or code from the user's perspective. Opening up the browser or program and navigating around in an attempt to test functionality and find bugs.

Automated Testing

Automated testing, on the other hand, is writing code that checks to see if other code works. Contrary to manual testing, the specifications remain constant from test to test. The biggest advantage is being able to test *many* things much faster.

Testing

It's the combination of these two testing techniques that will flush out as many bugs and unintended side-effects as possible, and ensure our program does what we say it will.

There are two main types of automated tests: Unit and End-to-End (E2E). E2E tests test an application as a whole. Unit tests test the smallest pieces of code, or units.

Testing Example

Purwadhika
Digital Technology School

- Install **Jest** in your computer globally with "***npm install jest -g***".
- Make a file named "**sum.js**" with code like this :

```
function sum(a, b) {
  return a + b;
}

module.exports = sum;
```

Testing Example

Purwadhika
Digital Technology School

- Make a file named “**sum.spec.js**” with code like this (we have to name the file with **.spec** or **.test** so that it can be tested by **jest**).
- Create **jest** configuration file named “**config.json**”. Fill it with an empty object first. You can check all the configuration here.
- Run “**jest --config=config.json**” in your project directory.

```
const sum = require("./sum.js");

test("Testing sum function", () => {
  expect(sum(1, 2)).toBe(3);
});
```

```
PASS ./sum.spec.js
  √ Testing sum function (3 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.931 s
Ran all test suites.
```

Explanation

Purwadhika
Digital Technology School

```
function sum(a, b) {
  return a + b;
}

module.exports = sum;
```

```
const sum = require("./sum.js");

test("Testing sum function", () => {
  expect(sum(1, 2)).toBe(3);
});
```

What happened in code beside? We make a function named sum that return a + b.

To test with jest we use test function that receive couple of arguments. First argument is for the title or description testing, second argument is a function that contain assertion of our code.

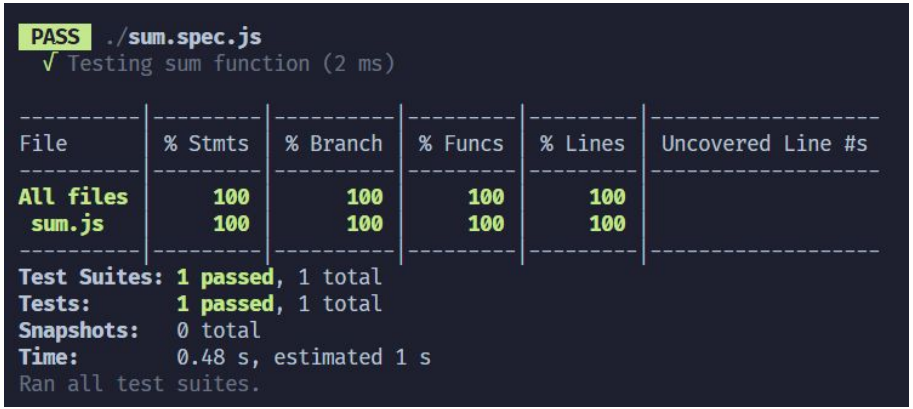
expect() is a function that receive a value, and toBe() is the result we expect from the value in expect().

sum(1, 2) surely will return 3, so our testing will success like we have tried before

Testing with Coverage

Purwadhika
Digital Technology School

- If we want to do more in-depth testing, we can add “**collectCoverage**” in configuration file.
- If we run the test again, the result will be different from the previous one.
- You can see that the result will inform us about statements, branch, functions and lines that tested by **jest**.
- It will also generate a folder named "**coverage**" containing the test result and coverage.



Testing with Coverage

Purwadhika
Digital Technology School

As example, if we add one more function in sum.js, but we don't test that function, then the coverage will tell us that there is a function that is not tested.

```
PASS ./sum.spec.js
  √ Testing sum function (1 ms)

----------|---------|----------|---------|---------|-------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------
All files |   66.66 |      100 |      50 |   66.66 |
 sum.js   |   66.66 |      100 |      50 |   66.66 | 6
----------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.678 s, estimated 1 s
Ran all test suites.
```

```
function sum(a, b) {
  return a + b;
}

// not exported and not tested
function average(numbers) {
  return 0;
}

module.exports = sum;
```

Testing

Purwadhika
Digital Technology School

There are many more things that can be done by jest besides checking the expected value, we can see other commands in the official documentation from Jest.
You can check it here.

Exercise

Purwadhika
Digital Technology School

- Create unit test from all your exercise before.

Thank You!

Purwadhika
Digital Technology School