Web Advanced

Jest

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Using Matchers

EDIT

Jest uses "matchers" to let you test values in different ways. This document will introduce some commonly used matchers. For the full list, see the `expect` API doc.

Common Matchers

The simplest way to test a value is with exact equality.

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4);
});
```

In this code, `expect(2 + 2)` returns an "expectation" object. You typically won't do much with these expectation objects except call matchers on them. In this code, `.toBe(4)` is the matcher. When Jest runs, it tracks all the failing matchers so that it can print out nice error messages for you.

`toBe` uses `Object.is` to test exact equality. If you want to check the value of an object, use `toEqual` instead:

```
test('object assignment', () => {
  const data = {one: 1};
  data['two'] = 2;
  expect(data).toEqual({one: 1, two: 2});
});
```

`toEqual` recursively checks every field of an object or array.

You can also test for the opposite of a matcher:

```
test('adding positive numbers is not zero', () => {
  for (let a = 1; a < 10; a++) {
    for (let b = 1; b < 10; b++) {
      expect(a + b).not.toBe(0);
    }
  }
});
```

`.toThrow(error?)`

Also under the alias: `.toThrowError(error?)`

Use `.toThrow` to test that a function throws when it is called. For example, if we want to test that `drinkFlavor('octopus')` throws, because octopus flavor is too disgusting to drink, we could write:

```
test('throws on octopus', () => {
  expect(() => {
    drinkFlavor('octopus');
  }).toThrow();
});
```

> Note: You must wrap the code in a function, otherwise the error will not be caught and the assertion will fail.

```
import Point from '../../../src/js/drawing/Point';

test('constructor to generate a Point object',
    () => {
        let point = new Point(1,1);
        expect(point).toBeInstanceOf(Point);
    });

test('constructor to throw error if 1st param. not a number',
    () => {
        expect(() => {
            new Point("a", 1);
        }).toThrow(Error);
    });

test('constructor to throw error if 2nd parameter not a number',
    () => {
        expect(() => {
            new Point(1, "a");
        }).toThrow(Error);
    });
```

```
test('getX to return the correct value',
    () => {
        let point = new Point(1, 2);
        let x = point.getX();
        expect(x).toBe(1);
    });

test('toString to return the correct value', () => {
    let point=new Point(1,2);
    let returnedString = point.toString();
    expect(returnedString).toBe("(1,2)");
});
```

describe: de 3 tests van de constructor
worden hieronder samengevoegd

```
import Point from '../../../src/js/drawing/Point';
describe('constructor',
    () => {
        it('should generate a Point-object for valid args',
            () => {
                let point = new Point(1, 1);
                expect(point).toBeInstanceOf(Point)
            }
        )
        it('should throw error when 1st parameter is not a number
            () => {
                expect(() => {
                    new Point("a", 1);
                }).toThrow(Error)
            })
        it('should throw error when 2nd parameter is not a number
            () => {
                expect(() => {
                    new Point(1, "a");
                }).toThrow(Error)
            })
    }
);
```

- `toBe` compares strict equality, using `===`
- `toEqual` compares the values of two variables. If it's an object or array, checks equality of all the properties or elements
- `toBeNull` is true when passing a null value
- `toBeDefined` is true when passing a defined value (opposite as above)
- `toBeUndefined` is true when passing an undefined value
- `toBeCloseTo` is used to compare floating values, avoid rounding errors
- `toBeTruthy` true if the value is considered true (like an `if` does)
- `toBeFalsy` true if the value is considered false (like an `if` does)
- `toBeGreaterThan` true if the result of expect() is higher than the argument
- `toBeGreaterThanOrEqual` true if the result of expect() is equal to the argument, or higher than the argument
- `toBeLessThan` true if the result of expect() is lower than the argument
- `toBeLessThanOrEqual` true if the result of expect() is equal to the argument, or lower than the argument
- `toMatch` is used to compare strings with regular expression pattern matching
- `toContain` is used in arrays, true if the expected array contains the argument in its elements set
- `toHaveLength(number)`: checks the length of an array
- `toHaveProperty(key, value)`: checks if an object has a property, and optionally checks its value
- `toThrow` checks if a function you pass throws an exception (in general) or a specific exception
- `toBeInstanceOf()`: checks if an object is an instance of a class

MOCK A SINGLE FUNCTION

More simply, you can mock a single function using `jest.fn()`:

```
const mathjs = require('mathjs')

mathjs.log = jest.fn(() => 'test')
test(`The mathjs log function`, () => {
  const result = mathjs.log(10000, 10)
  expect(result).toBe('test')
  expect(mathjs.log).toHaveBeenCalled()
  expect(mathjs.log).toHaveBeenCalledWith(10000, 10)
})
```

You can also use `jest.fn().mockReturnValue('test')` to create a simple mock that does nothing except returning a value.

PRE-BUILT MOCKS

You can find pre-made mocks for popular libraries. For example this package https://github.com/jefflau/jest-fetch-mock allows you to mock `fetch()` calls, and provide sample return values without interacting with the actual server in your tests.