Unit Testing in JavaScript: Jest

05/02/2024

About us

Aday Cuesta Correa
(alu0101483887)

Juan Rodríguez Suárez
(alu0101477596)

index.html

index.css

Index {

01 History

02 What is Unit Testing?

03 How to install

04 Syntax and How to use

05 Matchers

06 Tips & Tricks

07 Bibliography

}

history.html

history.css

01 {

[History]

< How it all began >

}

Jest

history.html

history.css

The beginning of ‘Unit Testing’ {

> < For the first 50 years of computer history, **unit testing** and **debugging** were essentially the same thing >
>
> < **Kent Bent** created JUnit. He called the approach **unit test** >

}

< Kent Bent >

Jest

history.html

history.css

The evolution to ‘TDD’ {

> < The combination of **code refactoring** and **unit testing** led to **Test-Driven Development** >
>
> < In **TDD,** code must be testable before it is even created >

}

< Martin Fowler >

Jest

what_is_ut.html

what_is_ut.css

1
2
3
4
5
6
7
8
9
10
11
12
13
14

02 {

[What is Unit Testing?]

< How does it works? >

}

what_is_ut.html

what_is_ut.css

Software development methodology {

- Simple
- Quality code
- Supported by many languages

}

Jest

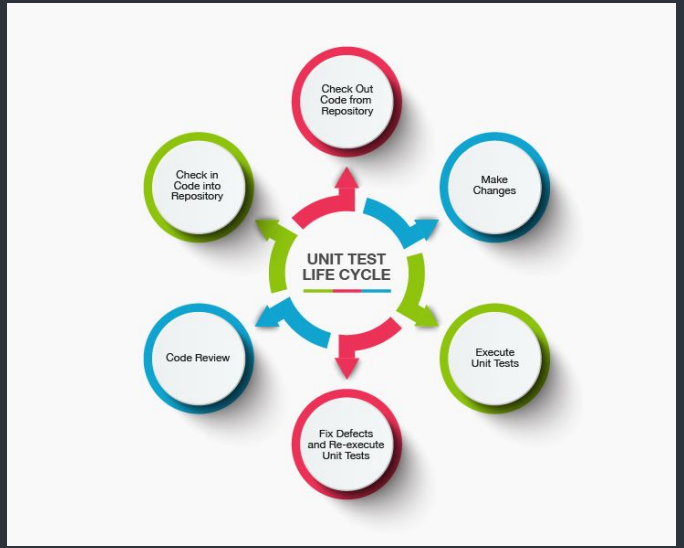
what_is_ut.html

what_is_ut.css

What is it about? {

1. Write your code
2. Write tests which verify certain functionalities of your code
3. Execute tests
4. Fix errors on your code and repeat

}

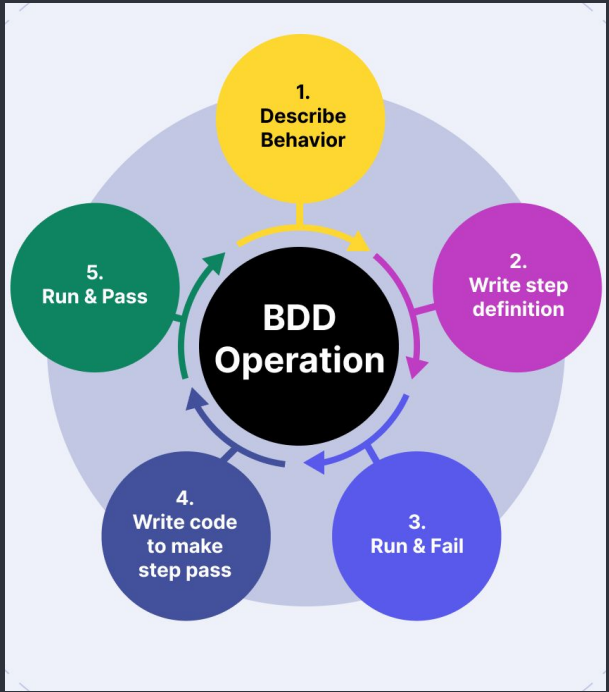


Jest

Other methodologies {

- Test Driven Development (TDD)
- Snapshot Testing (modern approach)
- Behavior-Driven Development, an extension of TDD (BDD)

}



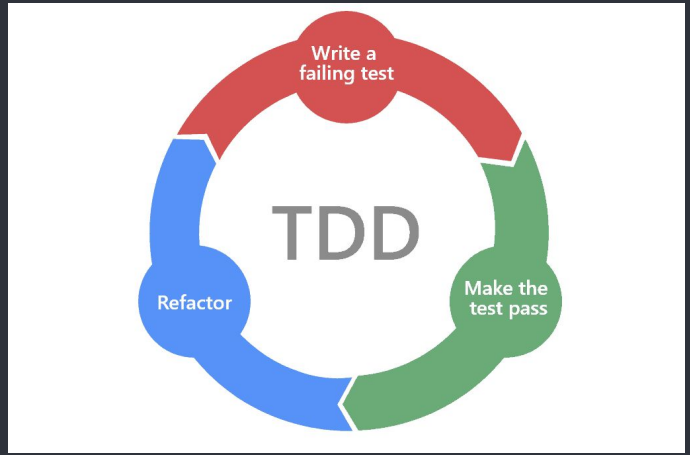
what_is_ut.html

what_is_ut.css

Approach to **TDD** {

- Code ***must be*** testable even ***before*** it's created

}



Jest

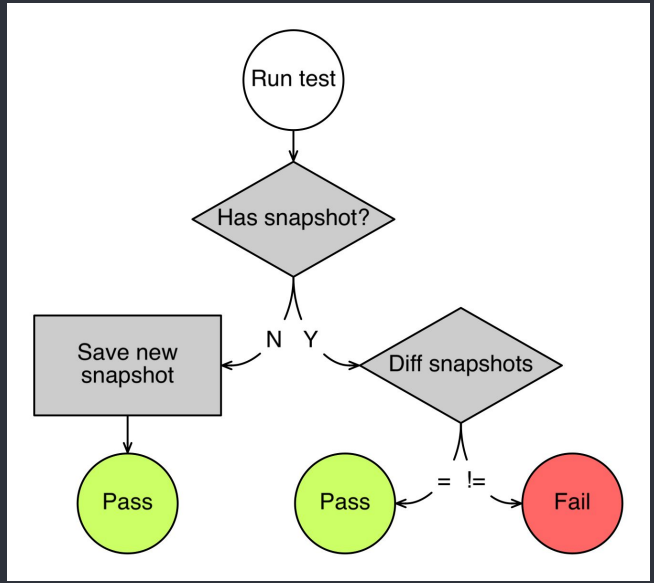
what_is_ut.html

what_is_ut.css

Modern approach of testing {

- Capturing code outputs (**snapshot**)
- Compare new outputs to the snapshot and see the differences

}



Jest

what_is_ut.html

what_is_ut.css

Advantages of UT {

- Early bug detection
- Eases code comprehension
- Safe refactoring of code
- Measure of quality

}

Disadvantages of UT {

- High initial cost
- Possible lack of code covering
- Sensitive to frequent code changes

- **UT does not increase development time if done well!**

}

what_is_ut.html

what_is_ut.css

Unit Testing frameworks {

- Python: PyUnit
- Java: JUnit o TestNG
- C#: NUnit
- Ruby: RSpec o Test::Unit (LPP)
- C++: Google Test (IB)
- **JavaScript**: Mocha o **Jest** (PAI)

}

Jest

installation.html

installation.css

```
03 {

  [Installation]

      < In few steps >

}
```

Jest

installation.html

installation.css

About Jest {

‘What is Jest?’

<Jest is a JavaScript testing framework designed
to ensure correctness of any JavaScript codebase >

‘Advantages/Features’

1•– Zero configuration needed;

2•– Fast;

3•– Built in code coverage;

4•– Isolated and Sandboxed tests;

5•– Support Snapshot Testing;

}

installation.html

installation.css

Jest Installation ‘Step by Step’ {

- Step 01 yarn add --dev jest
  - Step 02 **npm install --save-dev jest**
    - ‘What if I want to install the module globally?’
- Step 01 npm install -g jest

}

Installation of Jest In a Node-based project ‘Step by Step’ {

Step 01 Create a folder/directory with a name as your project name, for example → mkdir myFirstNodeProject

Step 02 cd myFirstNodeProject; npm init

Step 03 Keep Pressing Enter

Step 04 npm install --save-dev jest

Step 05 Configure the npm test script to run the Jest tests i.e. when the command ‘**npm test**’ is executed

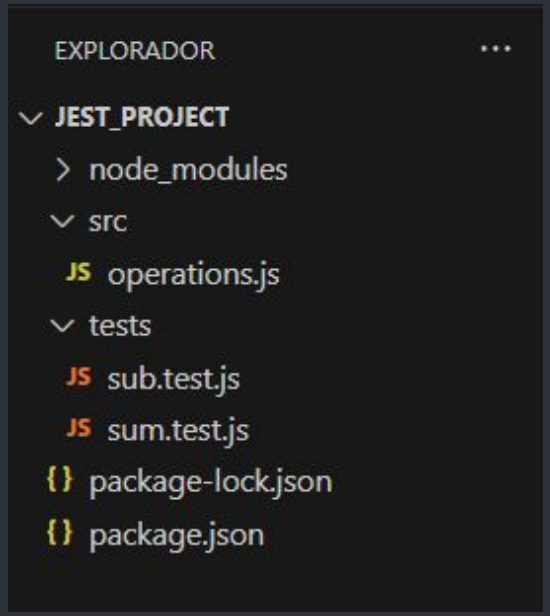
}

package.json {

```
"name": "jest-e2e",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
  "test": "jest"
},
"author": "",
"license": "ISC",
"dependencies": {
  "jest": "^25.1.0"
}
```

}

Directory structure {



1. All ‘useful’ code in ./src/ directory
2. All tests in ./tests/ directory
3. **Test files** names like ‘unit.test.js’

}

Note when using ES Modules {

- ES Modules are not the default option in Node.
- In order to use them instead of CommonJS, we need to change our package.json:

```
"type": "module",
▷ Depurar
"scripts": {
  "test": "node --experimental-vm-modules node_modules/jest/bin/jest.js"
},
```

}

syntax_usage.html

syntax_usage.css

```
04 {
    [Syntax and How to use]
        < With examples! >
}
```

Jest

describe keyword {

- Defines a group of related ***tests*** about some specific feature in our code.
- Takes two arguments:
  - String : *name* → Description of our group of tests.
  - Function : *fn* → Function which contains all of our ***tests***.
- Best practice: One ***describe*** per class or function

}

test keyword {

- Defines a group of ***expectations*** about an even more concrete aspect of our code described by the ***describe*** the test is surrounded by.
- Takes two arguments:
  - String : *name* → Description of our test.
  - Function : *fn* → Function which contains all of our ***expectations***.
  - Number : *timeout* → Maximum time (milliseconds) for a test to run (default 5 ms).
- Best practice: One ***test*** per functionality.

}

expect keyword {

- The **expect** function is used every time we want to test a value.
- We will use **expect** along with a **"matcher"** function to assert something about a value.
- Takes two arguments:
  - The value that your code produces.
  - Any argument to the matcher should be the correct value.
- Best practice: Use fairly simple expects so the code is easier to understand.

}

syntax_usage.html

syntax_usage.css

First we need some code {

```
'myFirstNodeProject/src/calculator.js'
const mathOperations = {
  sum: function(a,b) {
    return a + b;
  },
  diff: function(a,b) {
    return a - b;
  },
  product: function(a,b) {
    return a * b
  }
}
} module.exports = mathOperations
```

Jest

Let's create a test {

```
    'myFirstNodeProject/test/calculator.test.js'

     //This should be always at the top of the file
    //In this case we are using our calculator example code
    const mathOperations = require('./calculator');

    describe('Calculator tests', () => {
     test('adding 1 + 2 should return 3', () => {
       expect(mathOperations.sum(1, 2)).toBe(3);
     });
    })
}
```

syntax_usage.html

syntax_usage.css

Let's rewrite it {

```
'myFirstNodeProject/test/calculator.test.js'
const mathOperations = require('./calculator');

describe('Calculator tests', () => {
  test('adding 1 + 2 should return 3', () => {
    // arrange and act
    let result = mathOperations.sum(1, 2);
    // assert
    expect(result).toBe(3);
  });
})
```

}

Jest

syntax_usage.html

syntax_usage.css

Test output {

```
PASS  ./calculator.test.js
  Calculator tests
    ✓ adding 1 + 2 should return 3 (2ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.798s, estimated 1s
```

}

Jest

Let's create another test {

```
    'myFirstNodeProject/test/calculator.test.js'

    describe('Calculator tests', () => {
     test('adding 1 + 2 should return 10', () => {
       // arrange and act
       let result = mathOperations.sum(1,2);
       // assert
       expect(result).toBe(10);
     });
    })

}
```

syntax_usage.html

syntax_usage.css

Test output {

```
FAIL  ./calculator.test.js
 Calculator tests
   × adding 1 + 2 should return 10 (4ms)

 ● Calculator tests › adding 1 + 2 should return 10

   expect(received).toBe(expected) // Object.is equality

   Expected: 10
   Received: 3

      7 |
      8 |     // assert
    > 9 |     expect(result).toBe(10);
        |                    ^
     10 |   });
     11 | })
     12 |

     at Object.<anonymous> (calculator.test.js:9:20)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        0.846s, estimated 1s
```

}

Jest

syntax_usage.html

syntax_usage.css

One big describe block {

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
describe('Calculator tests', () => {
  test('adding 1 + 2 should return 3', () => {
    // arrange and act
    let result = mathOperations.sum(1,2)
    // assert
    expect(result).toBe(3);
  });
  test('subtracting 2 from 10 should return 8', () => {
    // arrange and act
    let result = mathOperations.diff(10,2)
    // assert
    expect(result).toBe(8);
  });
  test('multiplying 2 and 8 should return 16', () => {
    // arrange and act
    let result = mathOperations.product(2,8)
    // assert
    expect(result).toBe(16);
  });
})
```

}

Jest

syntax_usage.html

syntax_usage.css

Test output {

```
 PASS  ./calculator.test.js
  Calculator tests
    ✓ adding 1 + 2 should return 3 (2ms)
    ✓ subtracting 2 from 10 should return 8
    ✓ multiplying 2 and 8 should return 16 (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.834s, estimated 1s
Ran all test suites.
```

}

Jest

matchers.html

matchers.css

```
05 {
  [Matchers]
    < Almost all of them >
}
```

Jest

matchers.html

matchers.css

```
Equality Matchers {

  test('equality matchers', () => {
    expect(2 * 2).toBe(4);
    expect(4 - 2).not.toBe(1);
  })

}
```

Jest

matchers.html

matchers.css

Truthiness Matchers {

```
    test('truthy operators', () => {
        let name = 'Software testing help';
        let n = null;
        expect(n).toBeNull();
        expect(name).not.toBeNull;
        // name has a valid value
        expect(name).toBeTruthy();
        //pass - as null is non success
        expect(n).not.toBeTruthy();
        // pass - null treated as false or negative
        expect(n).toBeFalsy();
        // 0 - treated as false
        expect(0).toBeFalsy();
    })
}
```

Jest

matchers.html

matchers.css

Number Matchers {

```
  test('numeric operators', () => {
    let num1 = 100;
    let num2 = -20;
    let num3 = 0;
    // greater than
    expect(num1).toBeGreaterThan(10)
    // less than or equal
    expect(num2).toBeLessThanOrEqual(0)
    // greater than or equal
    expect(num3).toBeGreaterThanOrEqual(0)
  })

}
```

matchers.html

matchers.css

String Matchers {

```
test('string matchers',() => {
  let string1 = 'software testing help';
  // test for success match
  expect(string1).toMatch(/test/);
  // test for failure match
  expect(string1).not.toMatch(/abc/);
})
```

}

matchers.html

matchers.css

Floating Point Matchers {

```
1
2
3
4
5     test('adding works sanely with decimals', () => {
6       let float1 = 0.2;
7       let float2 = 0.1;
8       expect(float1 + float2).toBeCloseTo(0.3, 5);
9     })
10
11
12
13
14  }
```

matchers.html

matchers.css

toHaveBeenCalled Matcher {

```
  function drinkAll(callback, flavour) {
    if (flavour !== 'octopus') {
      callback(flavour);
    }
  }

  test('drinks something lemon-flavoured', () => {
    const drink = jest.fn(); // Spy function
    drinkAll(drink, 'lemon');
    expect(drink).toHaveBeenCalled();
  });
}
```

matchers.html

matchers.css

toHaveBeenCalledTimes Matcher {

```
test('sum function is called twice', () ⇒ {
  jest.spyOn(mathOperations, 'sum'); // Spy the sum function

  mathOperations.sum(2, 3);
  mathOperations.sum(4, 5);

  expect(mathOperations.sum).toHaveBeenCalledTimes(2);
});
```

}

matchers.html

matchers.css

toHaveBeenCalledWith Matcher {

```
  test('sum function is called with specific arguments', () => {
    jest.spyOn(mathOperations, 'sum');

    mathOperations.sum(2, 3);

    expect(mathOperations.sum).toHaveBeenCalledWith(2, 3);

    mathOperations.sum(4, 5);

    expect(mathOperations.sum).toHaveBeenCalledWith(4, 5);
  });
}
```

matchers.html

matchers.css

toHaveReturned Matcher {

```
test('sum function is called and has returned a value', () ⇒ {
  jest.spyOn(mathOperations, 'sum');
  mathOperations.sum(2, 3);
  expect(mathOperations.sum).toHaveReturned();
});
```

}

matchers.html

matchers.css

toHaveReturnedTimes Matcher {

```
  test('sum returns twice', () => {
    jest.spyOn(mathOperations, 'sum');

    mathOperations.sum.mockClear();

    mathOperations.sum(2, 3);
    mathOperations.sum(5, 4);

    expect(mathOperations.sum).toHaveReturnedTimes(2);
  });

}
```

matchers.html

matchers.css

toHaveReturnedWith Matcher {

```
test('sum function is called and returns a specific value', () ⇒ {
  jest.spyOn(mathOperations, 'sum');
  const result = mathOperations.sum(3, 4);
  expect(mathOperations.sum).toHaveReturnedWith(7);
});
```

}

matchers.html

matchers.css

toHaveLength Matcher {

```
test('arrayExample has a length of 5', () ⇒ {
  const arrayExample = [1, 2, 3, 4, 5];
  expect(arrayExample).toHaveLength(5);
});
```

}

matchers.html

matchers.css

toHaveProperty Matcher {

```
  test('carObject has the expected properties', () => {
    const carObject = {
      model: 'Corolla',
      features: {
        airConditioning: true
      },
    };
    expect(carObject).toHaveProperty('model');
    expect(carObject).toHaveProperty('features.airConditioning', true);
  });

}
```

matchers.html

matchers.css

toBeDefined Matcher {

```
test('exampleObject properties are defined', () ⇒ {
  const exampleObject = {
    property1: 'Hello'
  };
  expect(exampleObject.property1).toBeDefined();
});
```

}

matchers.html

matchers.css

toBeUndefined Matcher {

```
test('expecting undefined to be undefined', () ⇒ {
  expect(undefined).toBeUndefined();
})
```

}

matchers.html

matchers.css

```
toBeFalsy Matcher {

  test('falsyExample properties are falsy', () => {
    const falsyExample = {
      falsyProperty: 0,
      truthyProperty: 'Hello'
    };
    expect(falsyExample.falsyProperty).toBeFalsy();
    expect(falsyExample.truthyProperty).not.toBeFalsy();
  });


}
```

matchers.html

matchers.css

toBeInstanceOf Matcher {

```
test('isIntance of a class', () ⇒ {
  class A {}

  expect(new A()).toBeInstanceOf(A);
  expect(() ⇒ {}).toBeInstanceOf(Function);
});
```

}

matchers.html

matchers.css

toBeNaN Matcher {

```
test('passes when value is NaN', () ⇒ {
  expect(NaN).toBeNaN();
  expect(1).not.toBeNaN();
});
```

}

Jest

matchers.html

matchers.css

toContain Matcher {

```
test('arrayExample contains specific items', () ⇒ {
  const arrayExample = ['apple', 'banana', 'orange'];
  expect(arrayExample).toContain('banana');
  expect(arrayExample).not.toContain('kiwi');
});
```

}

matchers.html

matchers.css

toEqual Matchers {

```
test('2 variables are equal', () ⇒ {
  const number1 = 10;
  const number2 = number1;
  expect(number1).toEqual(number2);
})
```

}

matchers.html

matchers.css

toStrictEqual Matcher {

```
  class LaCroix {
    constructor(flavor) {
      this.flavor = flavor;
    }
  }

  describe('the La Croix cans on my desk', () => {
    test('are not semantically the same', () => {
      expect(new LaCroix('lemon')).toEqual({flavor: 'lemon'});
      expect(new LaCroix('lemon')).not.toStrictEqual({flavor: 'lemon'});
    });
  });

}
```

matchers.html

matchers.css

Exception Matchers {

```
function throwErrorExample() {
  throw new Error('This is a custom error message');
}

test('throwErrorExample throws an error', () => {
  expect(() => throwErrorExample()).toThrow();
});
```

}

tips_and_tricks.html

tips_and_tricks.css

```
06 {

    [Tips & Tricks]

        < Fancy ways to display your tests>

}
```

Jest Hooks {

- Special functions which execute in certain circumstances.
- They must be written inside a ***describe***, therefore, they only work inside that ***describe***.
- In Jest, we have four hooks:
  - beforeAll()
  - afterAll()
  - beforeEach()
  - afterEach()
- A hook takes a function as a parameter.

}

beforeAll & afterAll {

- Both functions will execute one time per ***describe***.
- **beforeAll** always executes first inside a ***describe***.
- **afterAll** always executes last inside a ***describe***.
- **beforeAll** is useful for setting up our tests environment (shared variables between tests ,etc.)
- **afterAll** is often used for cleaning up purposes (less used).

}

beforeAll & afterAll

```
describe('database connection', () => {
  let db;
  beforeAll(() => {
    db = new Database();
    db.connect();
  });

  afterAll(() => {
    db.disconnect();
  });

  test('database should be connected', () => {
    expect(db.isConnected).toBe(true);
  });

  test('example test', () => {
    expect(db.someMethod()).toBe(someExpectedValue);
  });
});
```

beforeEach & afterEach {

- Both functions will execute one time per ***test***.
- **beforeEach** always executes first inside a ***test***.
- **afterEach** always executes last inside a ***test***.
- Both of them are useful for testing container data structures due to the frequent insertion or deletion of elements inside it.

}

beforeEach & afterEach

```
describe('list manager', () => {
  let list;
  beforeEach(() => {
    list = new ListManager();
    list.addItem('item1');
    list.addItem('item2');
  });

  afterEach(() => {
    list.clearList();
  });

  test('addItem should add an item to the list', () => {
    list.addItem('item3');
    expect(list.items).toEqual(['item1', 'item2', 'item3']);
  });

  test('removeItem should remove an item from the list', () => {
    list.removeItem('item2');
    expect(list.items).toEqual(['item1']);
  });
});
```

Jest HTML Reporter {

- Console is fine, but not very human-friendly (particularly in 2024).
- Show your test results in a kind and cute website.
- Very easy to setup.

- **Installation**:

  npm install --save-dev jest-html-reporter

}

Jest HTML Reporter {

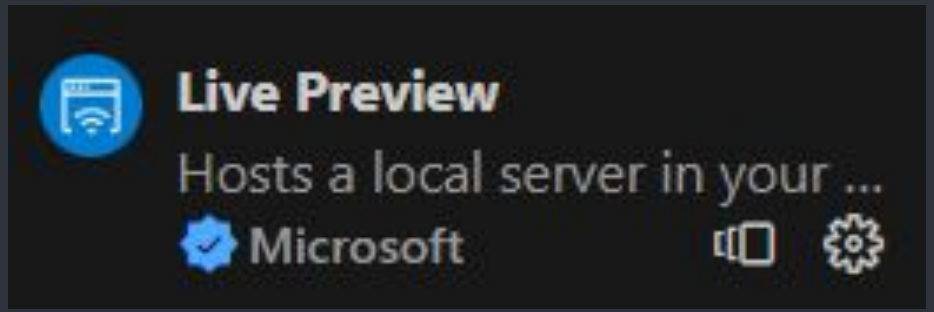
- Add the reporter to the Jest configuration in package.json:

```
"jest": {
  "reporters": [
    "default",
    [
      "./node_modules/jest-html-reporter",
      {
        "pageTitle": "My Amazing Test Report"
      }
    ]
  ]
}
```

}

Jest HTML Reporter {

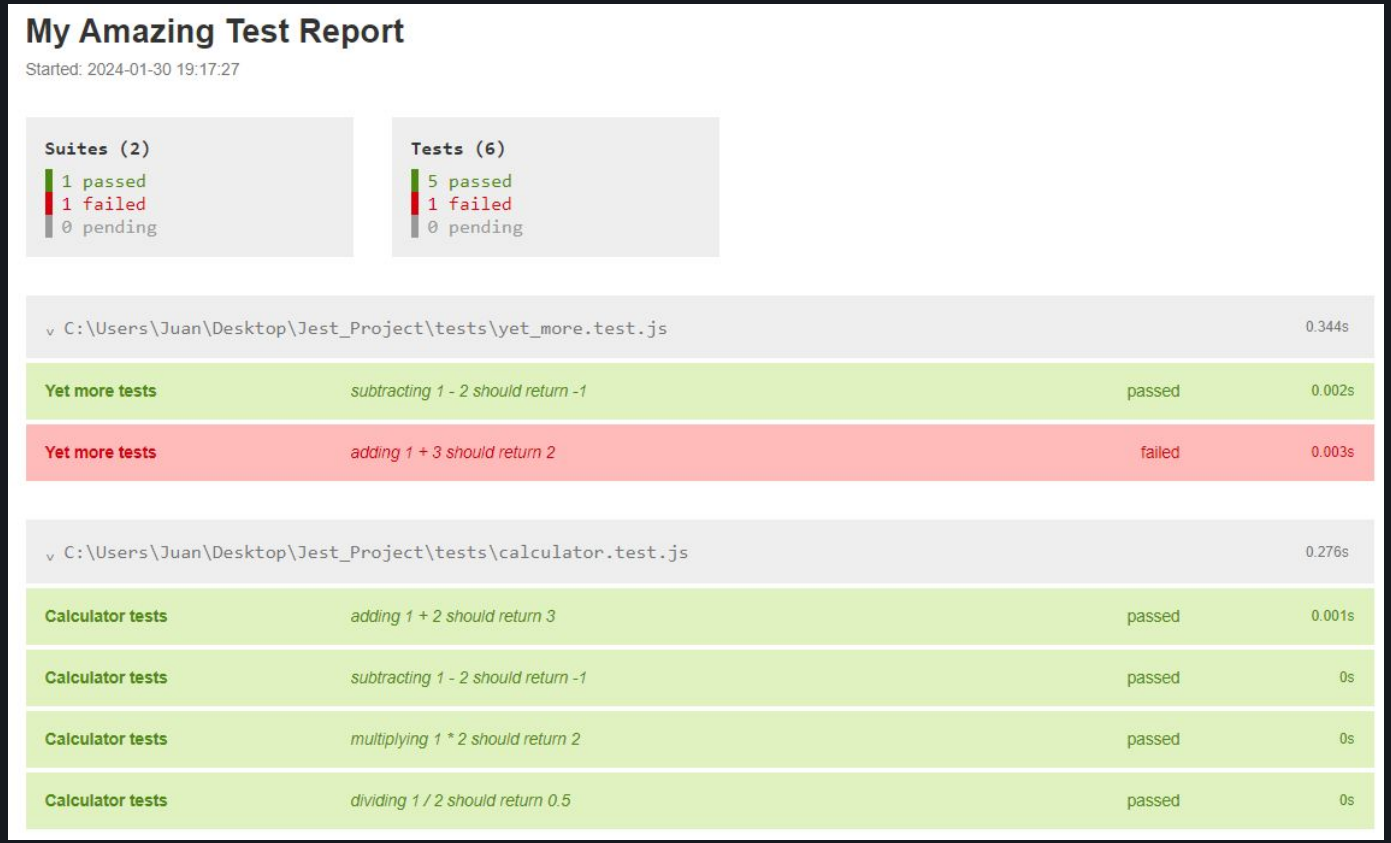
- Now, everytime you execute your tests (*npm test*) a *test-report.html* file will be created.
- In VSC, we recommend Live Preview extension made by MS to show HTML files easily.
- Right Click → Show Preview.
  - Available now in either VSC or http://localhost:3000/test-report.html



}

tips_and_tricks.html tips_and_tricks.css

Jest HTML Reporter



My Amazing Test Report

Started: 2024-01-30 19:17:27

Suites (2)
1 passed
1 failed
0 pending

Tests (6)
5 passed
1 failed
0 pending

v C:\Users\Juan\Desktop\Jest_Project\tests\yet_more.test.js 0.344s

Yet more tests	*subtracting 1 - 2 should return -1*	passed	0.002s
Yet more tests	*adding 1 + 3 should return 2*	failed	0.003s

v C:\Users\Juan\Desktop\Jest_Project\tests\calculator.test.js 0.276s

Calculator tests	*adding 1 + 2 should return 3*	passed	0.001s
Calculator tests	*subtracting 1 - 2 should return -1*	passed	0s
Calculator tests	*multiplying 1 * 2 should return 2*	passed	0s
Calculator tests	*dividing 1 / 2 should return 0.5*	passed	0s

Code coverage report {

- One of the most important metrics from a unit testing perspective.
- Measures what percentage of statements/branches are covered for the application under test.
- Even easier setup for Jest.
- **Installation**: Enable coverage report for Jest in package.json.

```
"jest": {
  "collectCoverage": true
}
```

}

tips_and_tricks.html tips_and_tricks.css

Code coverage report

```
PASS tests/yet_more.test.js
PASS tests/calculator.test.js
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
operations.js	100	100	100	100	
yet_more_operations.js	100	100	100	100	

tips_and_tricks.html tips_and_tricks.css

Code coverage report

```
PASS tests/yet_more.test.js
PASS tests/calculator.test.js
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	88.88	100	80	88.88	
operations.js	100	100	100	100	
yet_more_operations.js	50	100	0	50	3

bibliography.html

bibliography.css

```
07 {
    [Bibliography]
        < Our sources >
}
```

bibliography.html bibliography.css

Bibliography

- History: https://www.techtarget.com/searchsoftwarequality/answer/Is-unit-testing-an-important-aspect-of-software-development
- Advantages and Disadvantages: https://en.wikipedia.org/wiki/Unit_testing
- Unit Testing: https://testsigma.com/blog/unit-testing
- General: https://www.turing.com/kb/detailed-guide-on-unit-tests-and-advantages
- Jest: https://www.softwaretestinghelp.com/jest-testing-tutorial
- Documentation: https://jestjs.io/docs

thanks.html

thanks.css

Thanks for Watching!! {

You can ask us any questions right now or here:

alu01014838887@ull.edu.es

alu01014775596@ull.edu.es

}