Unit Testing in JavaScript: Jest

05/02/2024

About us

Aday Cuesta Correa
(alu0101483887)

Juan Rodríguez Suárez
(alu0101477596)

Index {

}

```
01 {

   [History]

      < How it all began >

}
```

The beginning of ‘Unit Testing’ {

< **Unit testing** and **debugging** were essentially the same thing >

< Code became so much **complex** >

< **Kent Bent** created JUnit. He called the approach **unit test** >

}

< Kent Bent >

The evolution to ‘TDD’ {

> < **Refactoring** suggests ways to transform code to make it more isolated and testable. >
>
> < In **TDD,** code must be testable before it is even created >

}

< Martin Fowler >

```
02 {

  [What is Unit Testing?]

     < How does it works? >

}
```

Software development methodology {

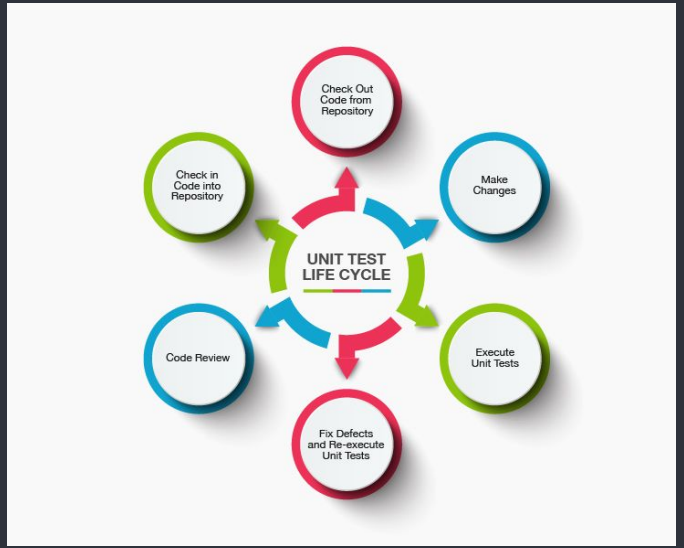
- Simple
- Quality code
- Supported by many languages

}

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

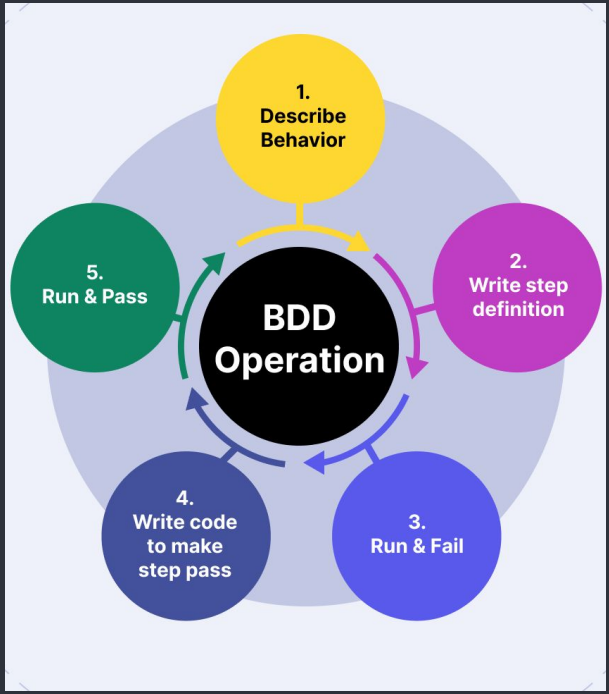
}



Other methodologies {

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

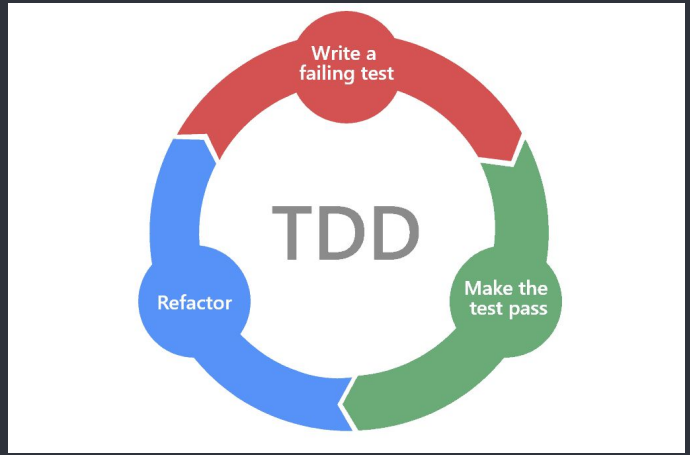
}



Approach to **TDD** {

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

}



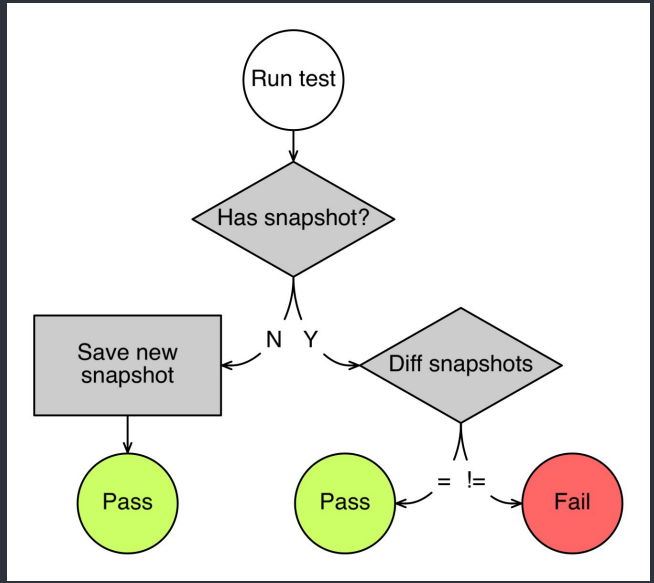
what_is_ut.html

what_is_ut.css

Modern approach of testing {

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

}



Advantages of UT {

- Early bug detection
- Eases code comprehension
- 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!**

}

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)

}

```
03 {

  [Installation]

      < In few steps >

}
```

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;

}

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

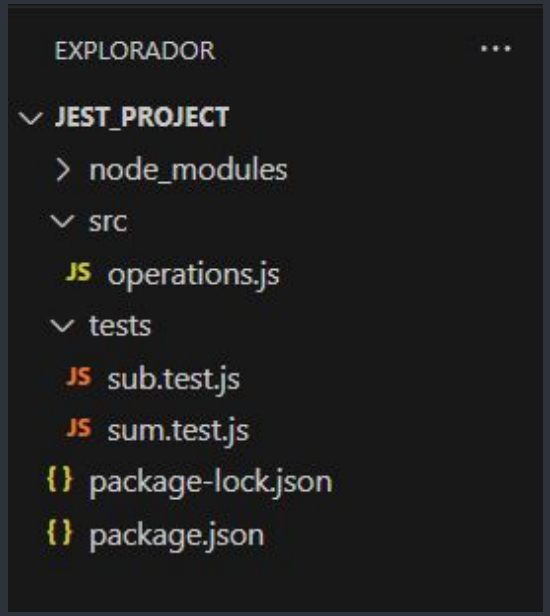
}

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"
},
```

}

```
04 {

    [Syntax and How to use]

        < With examples! >

}
```

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.

}

First we need some code {

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

}

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);
 });
})
}
```

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
        const RESULT = mathOperations.sum(1, 2);
        // assert
        expect(RESULT).toBe(3);
      });
  })
}
```

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
```

}

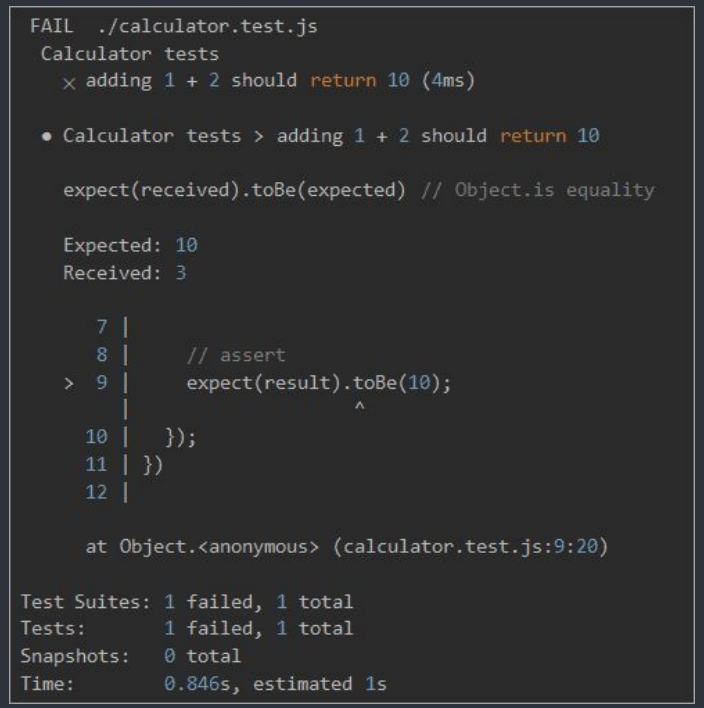
Let’s create another test {

```
    ‘myFirstNodeProject/test/calculator.test.js’

    describe(‘Calculator tests’, () ⇒ {
     test('adding 1 + 2 should return 10', () ⇒ {
       // arrange and act
       const RESULT = mathOperations.sum(1, 2);
       // assert
       expect(RESULT).toBe(10);
     });
    })

}
```

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

}

One big describe block {

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

}

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.
```

}

```
05 {

   [Matchers]

      < Almost all of them >

}
```

```
Equality Matchers {

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

}
```

Hello World

Poetry Club

Exception Matchers {

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

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

Collatz Conjecture

Bank Account

}

Truthiness Matchers {

```
    test('truthy operators', () => {
      const NAME = 'Software testing help';
      const SOME_VALUE = null;
      expect(SOME_VALUE).toBeNull();
      expect(NAME).not.toBeNull;
      expect(NAME).toBeTruthy();
      expect(SOME_VALUE).not.toBeTruthy();
      expect(SOME_VALUE).toBeFalsy();
      expect(0).toBeFalsy();
     })

        Pangram
}
```

Floating Point Matchers {

```
test('adding works sanely with decimals', () => {
  const FLOAT_1 = 0.2;
  const FLOAT_2 = 0.1;
  expect(FLOAT_1 + FLOAT_2).toBeCloseTo(0.3, 5);
})
```


}

String Matchers {

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

Parse String

}

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();
  });
}
```

Fruit Picker

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);
  });

}
```

Fruit Picker

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);
  });
}
```

Fruit Picker

Number Matchers {

```
  test('numeric operators', () => {
    const FIRST_NUMBER = 100;
    const SECOND_NUMBER = -20;
    const THIRD_NUMBER = 0;
    // greater than
    expect(FIRST_NUMBER).toBeGreaterThan(10)
    // less than or equal
    expect(SECOND_NUMBER).toBeLessThanOrEqual(0)
    // greater than or equal
    expect(THIRD_NUMBER).toBeGreaterThanOrEqual(0)
  })

}
```

toHaveReturned Matcher {

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

}

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);
  });
```

}

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);
});
```

}

toHaveLength Matcher {

```
1
2
3
4
5
6   test('arrayExample has a length of 5', () ⇒ {
7     const ARRAY_EXAMPLE = [1, 2, 3, 4, 5];
8     expect(ARRAY_EXAMPLE).toHaveLength(5);
9   });
10
11
12
13
14  }
```

toHaveProperty Matcher {

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

}
```

matchers.html

matchers.css

toBeDefined Matcher {

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

}

toBeUndefined Matcher {

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

}

toBeInstanceOf Matcher {

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

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

}

toBeNaN Matcher {

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

}

toContain Matcher {

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

}

You can check some examples on our GitHub! {

Just click the link:
https://github.com/ULL-ESIT-PAI-2023-2024/2023-2024-pai-testing-jest-juan-rs-aday-cc-ut_and_jest.git

}

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

```
4  describe('database connection', () => {
5    let database;
6    beforeAll(() => {
7      database = new Database();
8      database.connect();
9    });
10
11   afterAll(() => {
12     database.disconnect();
13   });
14
15   test('database should be connected', () => {
16     expect(database.isConnected).toBe(true);
17   });
18
19   test('example test', () => {
20     expect(database.someMethod()).toBe(someExpectedValue);
21   });
22 });
```

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 {

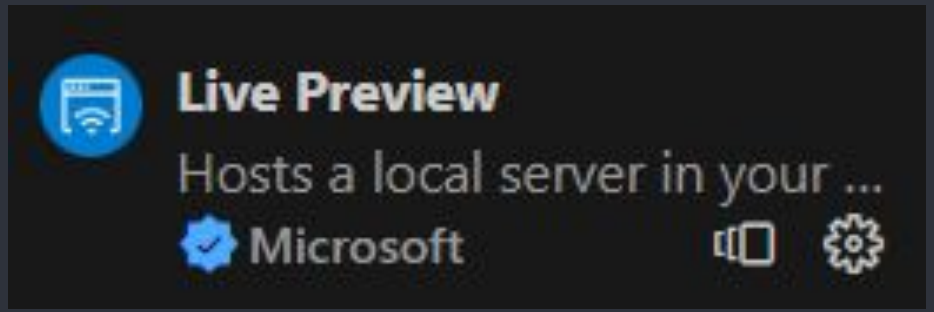
- Add the reporter to the Jest configuration in package.json:

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

}

Jest HTML Reporter {

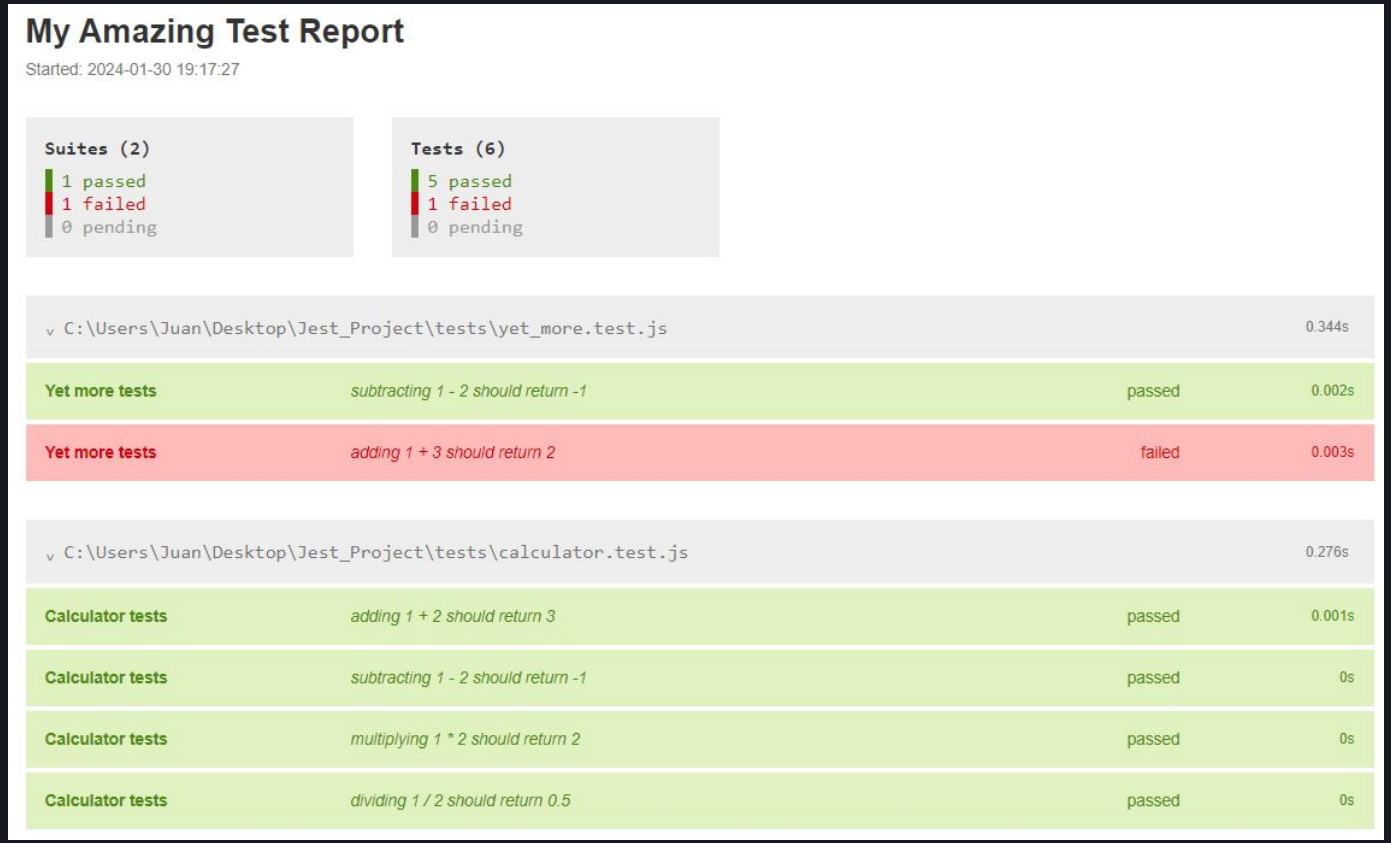
- Now, everytime you execute your tests (*npm test*) a *test-report.html* file will be created.
- In VSC, we recommend Live Preview extension made by MS to show HTML files easily.
- Right Click → Show Preview.
  - Available now in either VSC or http://localhost:3000/test-report.html



}

tips_and_tricks.html tips_and_tricks.css

Jest HTML Reporter



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

```
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
- Exercism: https://exercism.org/tracks/javascript
- Jutge: https://jutge.org

Thanks for Watching!! {

You can ask us any questions right now or here:

aday.cuesta.20@ull.edu.es

juan.suarez.12@ull.edu.es

}