Unit Testing in JavaScript: Jest

05/02/2024

About us

Aday Cuesta Correa
(alu0101483887)

Juan Rodríguez Suárez
(juan.suarez.12)

```
 1    Index {
 2
 3
 4        01 History
 5          02  What is Unit Testing?
 6
 7          03  How to install
 8            04  Syntax and How to use
 9
10            05  Matchers
11
12            06  Tips & Tricks
13              07  Bibliography
14    }
```

```
1
2       01 {
3
4
5
6          [History]
7
8
9              < How it all began >
10
11
12        }
13
14
```

1
2
3
4

The beginning of 'Unit Testing' {

5
6
7

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

8

< Code became so much **complex** >

9
10

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

11
12
13

}

14

< Kent Bent >

```
1
2     The evolution to
3     'TDD' {
4
5
6        < Refactoring suggests ways to
7        transform code to make it more
8        isolated and testable. >
9
10       < In TDD, code must be testable
11       before it is even created >
12
13    }
14
```

< Martin Fowler >

1
2   02 {
3
4
5
6   [What is Unit Testing?]
7
8   < How does it works? >
9
10
11
12   }
13
14

1
2
3 # Software
4 # development
5 # methodology {
6
7
8
9   - Simple
10   - Quality code
11   - Supported by many
12   languages
13
14 }

```
1
2
3
4
5    What is it about? {
6
7         1.  Write your code
8         2.  Write tests which verify
9             certain functionalities of
10            your code
11        3.  Execute tests
12        4.  Fix errors on your code and
              repeat
13   }
14
```

```
1
2  Other methodologies {
3
4          ● Test Driven
5            Development (TDD)
6
7          ● Snapshot Testing
8            (modern approach)
9
10         ● Behavior-Driven
11           Development, an
             extension of TDD (BDD)
12 }
13
14
```
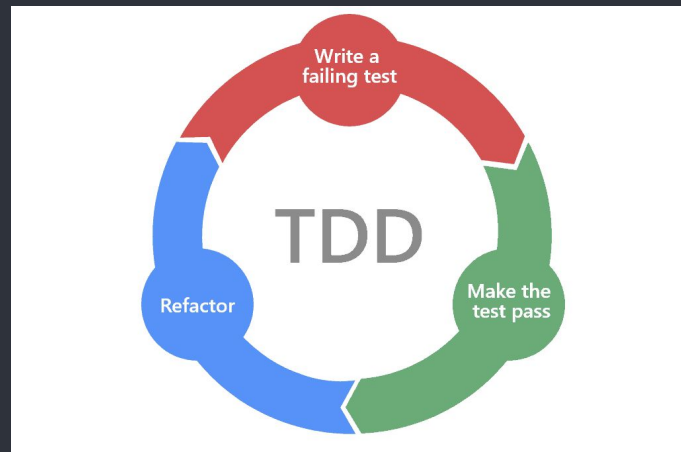
```
1
2
3
4
5    Approach to TDD {
6
7
8
9          ● Code must be testable
10             even before it's
11             created
12
13  }
14
```

```
1
2
3
4    Modern approach
5    of testing {
6
7
8        ● Capturing code outputs
9          (snapshot)
10       ● Compare new outputs to
11         the snapshot and see
12         the differences
13
14   }
```


Jest
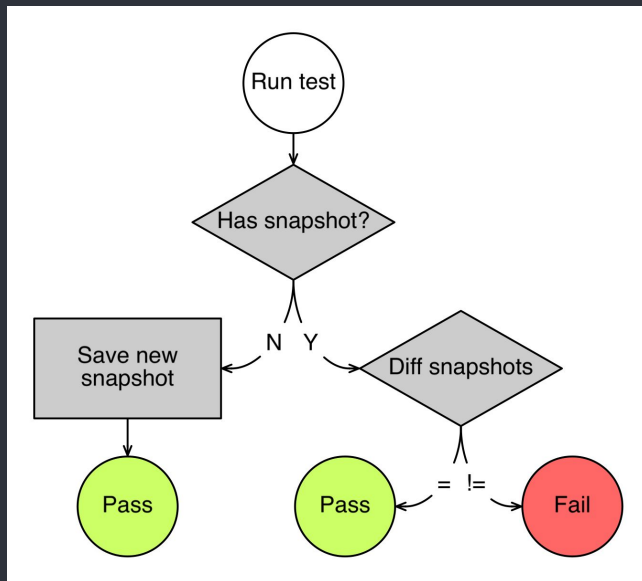
```
1   Advantages of UT {
2
3
4           ● Early bug detection
5           ● Eases code comprehension
6           ● Measure of quality
7
8
9
10
11
12
13
14   }
```

```
1    Disadvantages of UT {
2
3
4        ●   High initial cost
5        ●   Possible lack of code covering
6        ●   Sensitive to frequent code changes
7
8
9
10
11
12       ●   UT does not increase development time if done
13           well!
14   }
```

Jest

```
1  Unit Testing frameworks {
2
3          ●   Python: PyUnit
4
5          ●   Java: JUnit o TestNG
6          ●   C#: NUnit
7          ●   Ruby: RSpec o Test::Unit (LPP)
8          ●   C++: Google Test (IB)
9          ●   JavaScript: Mocha o Jest (PAI)
10
11
12
13
14  }
```

Jest

```
1
2     03 {
3
4
5
6     [Installation]
7
8
9          < In few steps >
10
11
12    }
13
14
```

```
1   About Jest {
2       'What is Jest?'
3
4           < Jest is a JavaScript testing framework designed
            to ensure correctness of any JavaScript codebase >
5
6       'Advantages/Features'
7           1•— Zero configuration needed;
8
9           2•— Fast;
10
11          3•— Built in code coverage;
12
13          4•— Isolated and Sandboxed tests;
            5•— Support Snapshot Testing;
14  }
```

Jest

```
1   Jest Installation 'Step by Step' {
2
3           Step 01  yarn add --dev jest
4
5
6             Step 02  npm install --save-dev jest
7
8
9                    'What if I want to install the module globally?'
10
11         Step 01  npm install -g jest
12
13
14  }
```

```
1    Installation of Jest In a Node-based
2    project 'Step by Step' {
3
4                    Step 01   Create a folder/directory with a name as your project name,
5                              for example → mkdir myFirstNodeProject
6                      Step 02   cd myFirstNodeProject; npm init
7
8                          Step 03   Keep Pressing Enter
9
10                           Step 04   npm install --save-dev jest
11                                              Configure the npm test script to
                                  Step 05   run the Jest tests i.e. when the
12                                            command 'npm test' is executed
13
14   }
```
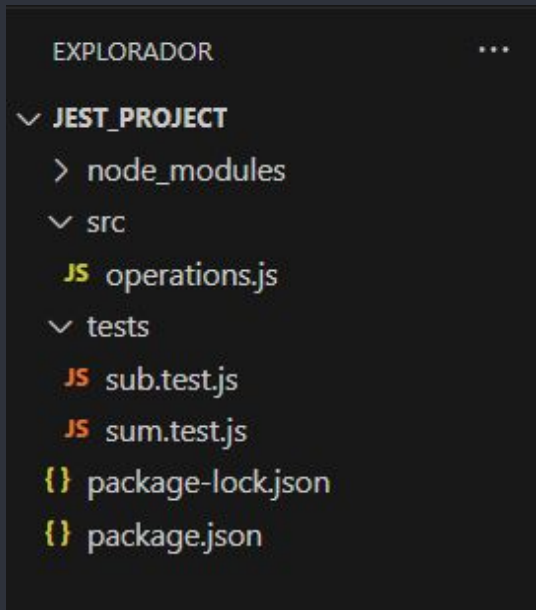
Jest

```
1   package.json {
2     "name": "jest-e2e",
3     "version": "1.0.0",
4     "description": "",
5     "main": "index.js",
6     "scripts": {
7       "test": "jest"
8     },
9     "author": "",
10    "license": "ISC",
11    "dependencies": {
12      "jest": "^25.1.0"
13    }
14  }
```

Jest

```
1    Directory structure {
2
3
4                                       1.  All 'useful' code in ./src/ directory
5                                       2.  All tests in ./tests/ directory
6                                       3.  Test files names like 'unit.test.js'
7
8
9
10
11
12
13
14   }
```

EXPLORADOR   ...

∨ JEST_PROJECT
  > node_modules
  ∨ src
    JS operations.js
  ∨ tests
    JS sub.test.js
    JS sum.test.js
  {} package-lock.json
  {} package.json

Jest

1 # Note when using ES Modules {
2
3   ● ES Modules are not the default option in Node.
4   ● In order to use them instead of CommonJS, we need
5   to change our package.json:
6
7
8
9
10
11
12
13
14 }

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

Jest

```
1
2      04 {
3
4
5
6        [Syntax and How to use]
7
8           < With examples! >
9
10
11
12      }
13
14
```

Jest

1 **describe** keyword {

2

3   ● Defines a group of related *tests* about some

4   specific feature in our code.

5   ● Takes two arguments:

6   ○ String : *name* → Description of our group of

           tests.

7   ○ Function : *fn* → Function which contains all

8   of our *tests*.

9   ● Best practice: One *describe* per class or function

10

11

12

13

14 }

Jest

```
1    test keyword {
2
3            ● Defines a group of expectations about an even more
4              concrete aspect of our code described by the
5              describe the test is surrounded by.
6            ● Takes two arguments:
7                ○ String : name → Description of our test.
8                ○ Function : fn → Function which contains all
8                  of our expectations.
9                ○ Number : timeout → Maximum time
10                 (milliseconds) for a test to run (default 5
                   ms).
11           ● Best practice: One test per functionality.
12
13
14   }
```

Jest

1 **expect** keyword {
2
3   ● The **expect** function is used every time we want to
4   test a value.
5   ● We will use **expect** along with a **"matcher"** function
6   to assert something about a value.
7   ● Takes two arguments:
8   ○ The value that your code produces.
9   ○ Any argument to the matcher should be the
                 correct value.
10   ● Best practice: Use fairly simple expects so the
11   code is easier to understand.
12
13
14 }

Jest

First we need some code {

```
1
2        'myFirstNodeProject/src/calculator.js'
3
4    const mathOperations = {
5      sum: function(first_number, second_number) {
6        return first_number + second_number;
7      },
8      diff: function(first_number, second_number) {
9        return first_number - second_number;
10     },
11     product: function(first_number, second_number) {
12       return first_number * second_number;
13     }
14   }
   } module.exports = mathOperations;
```

Jest

```
 1   Let's create a test {
 2
 3       'myFirstNodeProject/test/calculator.test.js'
 4
 5        //This should be always at the top of the file
 6       //In this case we are using our calculator example code
 7       const mathOperations = require('./calculator');
 8
 9       describe('Calculator tests', () ⇒ {
10         test('adding 1 + 2 should return 3', () ⇒ {
11           expect(mathOperations.sum(1, 2)).toBe(3);
12         });
13       })
14   }
```

Jest

```
1    Let's rewrite it {
2
3        'myFirstNodeProject/test/calculator.test.js'
4        const mathOperations = require('./calculator');
5
6        describe('Calculator tests', () ⇒ {
7          test('adding 1 + 2 should return 3', () ⇒ {
8            // arrange and act
9            const RESULT = mathOperations.sum(1, 2);
10           // assert
11           expect(RESULT).toBe(3);
12         });
13     })
14   }
```

Jest

```
1    Test output {
2
3
4
5
6
7
8
9
10
11
12
13
14    }
```

```
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 {

```
 1
 2
 3        'myFirstNodeProject/test/calculator.test.js'
 4
 5      describe('Calculator tests', () ⇒ {
 6       test('adding 1 + 2 should return 10', () ⇒ {
 7         // arrange and act
 8         const RESULT = mathOperations.sum(1, 2);
 9         // assert
         expect(RESULT).toBe(10);
10       });
11      })
12
13
14  }
```

Jest

```
1    Test output {
2
3
4
5
6
7
8
9
10
11
12
13
14    }
```

```
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

```
1   One big describe block {
2       describe('Calculator tests', () ⇒ {
3         test('adding 1 + 2 should return 3', () ⇒ {
4           // arrange and act
            const RESULT = mathOperations.sum(1,2)
5           // assert
            expect(RESULT).toBe(3);
6         });
7         test('subtracting 2 from 10 should return 8', () ⇒ {
            // arrange and act
8           const RESULT = mathOperations.diff(10,2)
            // assert
9           expect(RESULT).toBe(8);
10        });
11        test('multiplying 2 and 8 should return 16', () ⇒ {
            // arrange and act
12          const RESULT = mathOperations.product(2,8)
            // assert
13          expect(RESULT).toBe(16);
          });
14  }   })
```

Jest

```
1    Test output {
2
3        PASS   ./calculator.test.js
4         Calculator tests
5            ✓ adding 1 + 2 should return 3 (2ms)
6            ✓ subtracting 2 from 10 should return 8
7            ✓ multiplying 2 and 8 should return 16 (1ms)
8
9        Test Suites: 1 passed, 1 total
10       Tests:       3 passed, 3 total
11       Snapshots:   0 total
12       Time:        0.834s, estimated 1s
         Ran all test suites.
13
14   }
```

Jest

```
1
2      05 {
3
4
5
6        [Matchers]
7
8          < Almost all of them >
9
10
11
12     }
13
14
```

```
1   Equality Matchers {
2
3
4
5     test('equality matchers', () ⇒ {
6       expect(2 * 2).toBe(4);
7       expect(4 - 2).not.toEqual(1);
8     })
9
10
11
12          Hello World
13          Poetry Club
14  }
```

Jest

Exception Matchers {

```
1
2
3
4    function throwErrorExample() {
5      throw new Error('This is a custom error message');
6    }
7
8
9    test('throwErrorExample throws an error', () ⟹ {
10     expect(() ⟹ throwErrorExample()).toThrow();
11   });
12
13        Collatz Conjecture
14  }     Bank Account
```

Jest

37

Truthiness Matchers {

```
1
2        test('truthy operators', () ⟹ {
3            let name = 'Software testing help';
4            let some_value = null;
5            expect(some_value).toBeNull();
6            expect(name).not.toBeNull;
7            expect(name).toBeTruthy();
8            expect(some_value).not.toBeTruthy();
9            expect(some_value).toBeFalsy();
10           expect(0).toBeFalsy();
11       })
12
13              Pangram
14   }
```

Jest

```
1  Floating Point Matchers {
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      Freelancer Rates
13
14 }
```

Jest

```
1   String Matchers {
2
3
4       test('string matchers',() ⇒ {
5         const MY_SENTENCE = 'software testing help';
6         // test for success match
7         expect(MY_SENTENCE).toMatch(/test/);
8         // test for failure match
9
10        expect(MY_SENTENCE).not.toMatch(/abc/);
11      })
12
13        Parse String
14  }
```

toHaveBeenCalled Matcher {

```
 1
 2
 3      function drinkAll(callback, flavour) {
 4        if (flavour === 'octopus') {
 5          callback(flavour);
 6        }
 7      }
 8
 9      test('drinks something lemon-flavoured', () ⇒ {
10        const DRINK = jest.fn(); // Spy function
11        drinkAll(DRINK, 'lemon');
12        expect(DRINK).toHaveBeenCalled();
13      });
14   }
```

Fruit Picker

toHaveBeenCalledTimes Matcher {

```
 1
 2
 3
 4
 5    test('sum function is called twice', () ⇒ {
 6      jest.spyOn(mathOperations, 'sum'); // Spy the sum function
 7
 8      mathOperations.sum(2, 3);
 9      mathOperations.sum(4, 5);
10
11      expect(mathOperations.sum).toHaveBeenCalledTimes(2);
12    });
13
14 }

        Fruit Picker
```

toHaveBeenCalledWith Matcher {

```
1
2
3      test('sum function is called with specific arguments', () ⇒ {
4        jest.spyOn(mathOperations, 'sum');
5
6      mathOperations.sum(2, 3);
7
8        expect(mathOperations.sum).toHaveBeenCalledWith(2, 3);
9
10      mathOperations.sum(4, 5);
11
12        expect(mathOperations.sum).toHaveBeenCalledWith(4, 5);
13    });
14 }
         Fruit Picker
```

Jest

43

Number Matchers {

```
 1
 2
 3        test('numeric operators', () ⇒ {
 4          const FIRST_NUMBER = 100;
 5          const SECOND_NUMBER = -20;
 6          const THIRD_NUMBER = 0;
 7          // greater than
 8          expect(FIRST_NUMBER).toBeGreaterThan(10)
 9          // less than or equal
10          expect(SECOND_NUMBER).toBeLessThanOrEqual(0)
11          // greater than or equal
12          expect(THIRD_NUMBER).toBeGreaterThanOrEqual(0)
13        })
14  }
```

```
1   toHaveReturned Matcher {
2
3
4
5
6   test('sum function is called and has returned a value', () ⇒ {
7     jest.spyOn(mathOperations, 'sum');
8     mathOperations.sum(2, 3);
9     expect(mathOperations.sum).toHaveReturned();
10  });
11
12
13
14  }
```

toHaveReturnedTimes Matcher {

```
 1
 2
 3      test('sum returns twice', () ⇒ {
 4        jest.spyOn(mathOperations, 'sum');
 5
 6      mathOperations.sum.mockClear();
 7
 8      mathOperations.sum(2, 3);
 9      mathOperations.sum(5, 4);
10
11      expect(mathOperations.sum).toHaveReturnedTimes(2);
12    });
13
14  }
```

```
 1   toHaveReturnedWith Matcher {
 2
 3
 4
 5
 6   test('sum function is called and returns a specific value', () ⇒ {
 7     jest.spyOn(mathOperations, 'sum');
 8     const RESULT= mathOperations.sum(3, 4);
 9     expect(mathOperations.sum).toHaveReturnedWith(7);
10   });
11
12
13
14  }
```

```
1   toHaveLength Matcher {
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  }
```

Jest

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);
    });

}
```

toBeDefined Matcher {

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

}

```
1   toBeUndefined Matcher {
2
3
4
5    test('expecting undefined to be undefined', () ⇒ {
6      expect(undefined).toBeUndefined();
7    })
8
9
10
11
12
13
14  }
```

```
1    toBeInstanceOf Matcher {
2
3
4      test('isIntance of a class', () ⇒ {
5        class Coche {}
6
7        expect(new Coche()).toBeInstanceOf(Coche);
8        expect(() ⇒ {}).toBeInstanceOf(Function);
9      });
10
11
12
13
14    }
```

```
1   toBeNaN Matcher {
2
3
4     test('passes when value is NaN', () ⟹ {
5       expect(NaN).toBeNaN();
6       expect(1).not.toBeNaN();
7     });
8
9
10
11
12
13
14  }
```

Jest

```
 1   toContain Matcher {
 2
 3
 4    test('arrayExample contains specific items', () ⇒ {
 5      const ARRAY_EXAMPLE = ['apple', 'banana', 'orange'];
 6      expect(ARRAY_EXAMPLE).toContain('banana');
 7      expect(ARRAY_EXAMPLE).not.toContain('kiwi');
 8    });
 9
10
11
12
13
14  }
```

Jest

```
1    You can check some examples on our
2    GitHub! {
3
4
5
6        Just click the link:
7
8        https://github.com/ULL-ESIT-PAI-2023-2024/2023-2024
9        -pai-testing-jest-juan-rs-aday-cc-ut_and_jest.git
10
11
12
13
14   }
```

```
1
2      06 {
3
4
5
6        [Tips & Tricks]
7
8
9            < Fancy ways to display your tests>
10
11
12      }
13
14
```

Jest Hooks {

1
2
3   ● Special functions which execute in certain
4   circumstances.
5   ● They must be written inside a *describe*, therefore,
         they only work inside that *describe*.
6   ● In Jest, we have four hooks:
7   ○ beforeAll()
8   ○ afterAll()
9   ○ beforeEach()
10   ○ afterEach()
       ● A hook takes a function as a parameter.
11
12
13
14 }

1 **beforeAll & afterAll** {
2
3   ● Both functions will execute one time per *describe*.
4   ● **beforeAll** always executes first inside a *describe*.
5   ● **afterAll** always executes last inside a *describe*.
6   ● **beforeAll** is useful for setting up our tests
           environment (shared variables between tests ,etc.)
7   ● **afterAll** is often used for cleaning up purposes
8   (less used).
9
10
11
12
13
14 }

Jest

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   });
```

1 **beforeEach & afterEach** {
2
3   ● Both functions will execute one time per *test*.
4   ● **beforeEach** always executes first inside a *test*.
5   ● **afterEach** always executes last inside a *test*.
6   ● Both of them are useful for testing container data
7   structures due to the frequent insertion or
        deletion of elements inside it.
8
9
10
11
12
13
14 }

beforeEach & afterEach

```javascript
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']);
  });
});
```

61

1 **Jest HTML Reporter** {

2

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

7

8   ● **Installation**:

9

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

11

12

13

14 }

```
1   Jest HTML Reporter {
2
3       ● Add the reporter to the Jest configuration in
4           package.json:
5
6
7
8
9
10
11
12
13
14 }
```

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

1 **Jest HTML Reporter** {

2

3   ● Now, everytime you execute your tests (*npm test*) a
4   *test-report.html* file will be created.
5   ● In VSC, we recommend Live Preview extension made
      by MS to show HTML files easily.
6
7   ● Right Click → Show Preview.
        ○ Available now in either VSC or
8   http://localhost:3000/test-report.html

9

10

11

12

13

14 }
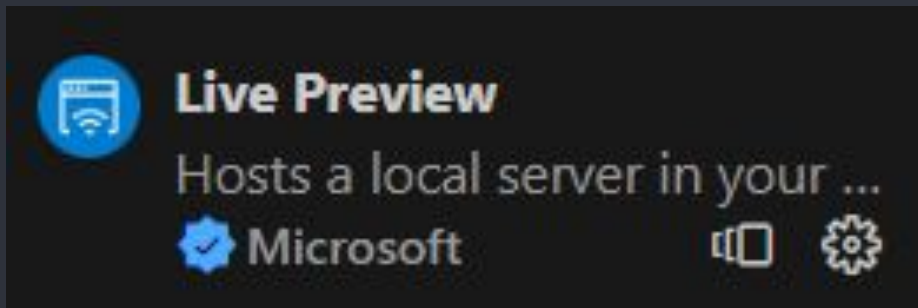
Jest HTML Reporter

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

∨ 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

∨ 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

65

1 # Code coverage report {

2

3   • One of the most important metrics from a unit
4   testing perspective.
5   • Measures what percentage of statements/branches
      are covered for the application under test.
6
7   • Even easier setup for Jest.
   • **Installation**: Enable coverage report for Jest in
8   package.json.

9

10

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

11

12

13

14 }

Jest

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  |
-----------------------|----------|----------|----------|----------|--------------------
```

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
---------------------------|---------|----------|---------|---------|-------------------
```

```
1
2      07 {
3
4
5
6        [Bibliography]
7
8          < Our sources >
9
10
11
12      }
13
14
```

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

1   **Thanks for Watching‼** {

2

3

4

5   You can ask us any questions right now or here:

6

7   aday.cuesta.20@ull.edu.es

8   juan.suarez.12@ull.edu.es

9

10

11

12

13

14   }

Jest