Testing JavaScript

Why Should I Test my Code?

- Adds Clarity
- Favors Decoupling
- Encourages Simplification
- Supports Extensibility

The Trouble with Tests

Brittle

Complex

Ignored

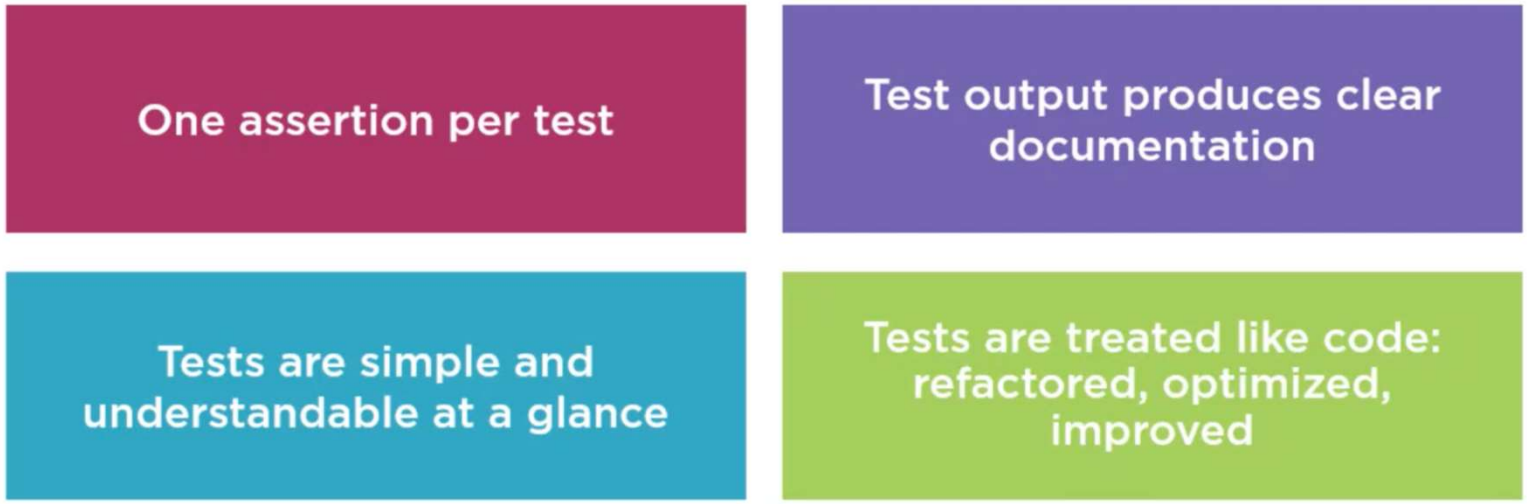
How to Make It Work?

Best Practices

Robust Tests

Do and Learn

Guidelines for Successful Unit Testing



How Do We Write Tough Tests?

The ends justify the means

Test what not how

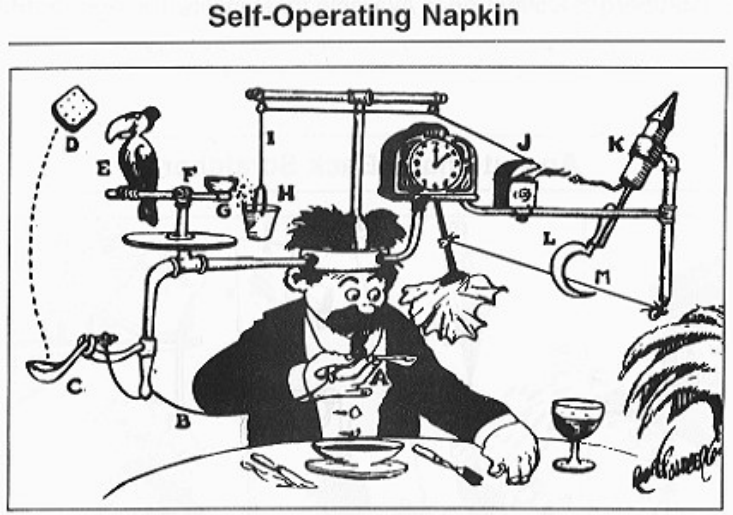
Less is more

Isolate and focus

Rube Goldberg

doing something simple in a very complicated way that is not necessary

Source: Merriam-Webster's Learner's Dictionary



Ask Yourself These Questions..

What does this function need to do?

What state needs to change?

What interactions need to happen?

What might change, and should it affect the test?

Tough Strategies

Use test variables, avoid literals

Use configuration values

Use nonsense or default literals for trivial values

S.O.L.I.D.

Inputs and Outputs

Test for Positive Results

Negative Results	Positive Results
Signal: No	Signal: Yes
Environment: Stable	Environment: Faulted
Outcome: Security	Outcome: Alerted
Action: None	Action: Fix the code

Take Care of False Results

False Negative Results

Signal: No

Environment: Faulted

Outcome: Defect unabated

Action: None taken

False Positive Results

Signal: Yes

Environment: Stable

Outcome: Awareness

Action: Fix or disable the test

Side Effects: Ignore positive results

Prevention: Tough tests

Mocking: Why and How

Why?

Use mocks and stubs to isolate. Use dynamic mocking frameworks to save time and simplify maintenance.

How?

Use stubs to enable state / output testing. Use mocks to test interactions. Do not test implementation details.

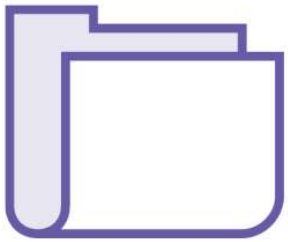
Why Mocking?

Reduce dependencies required by tests (faster execution)

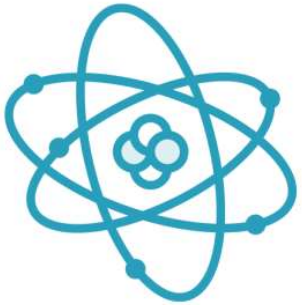
Prevent side-effects during testing

Build custom mocks to facilitate desired testing procedures

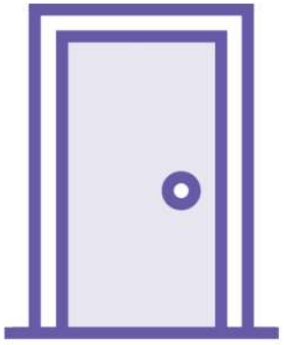
What Is a Mock?



A convincing duplicate of an object with no internal workings



Can be automatically or manually created



Has same API as original, but no side-effects

Spies and other mock features simplify testing

Make Your Mocks Tough

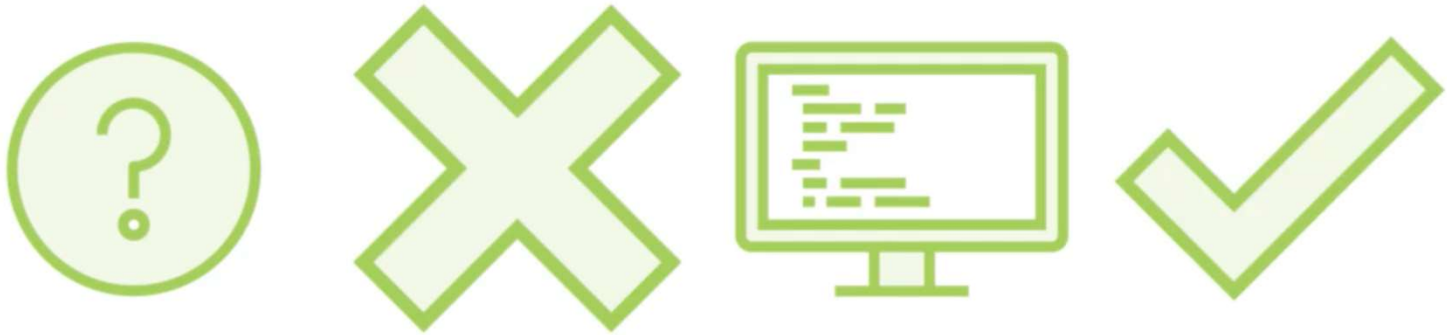
The Mocking Process

Scan the original object for methods, give the new object spy methods with the same names

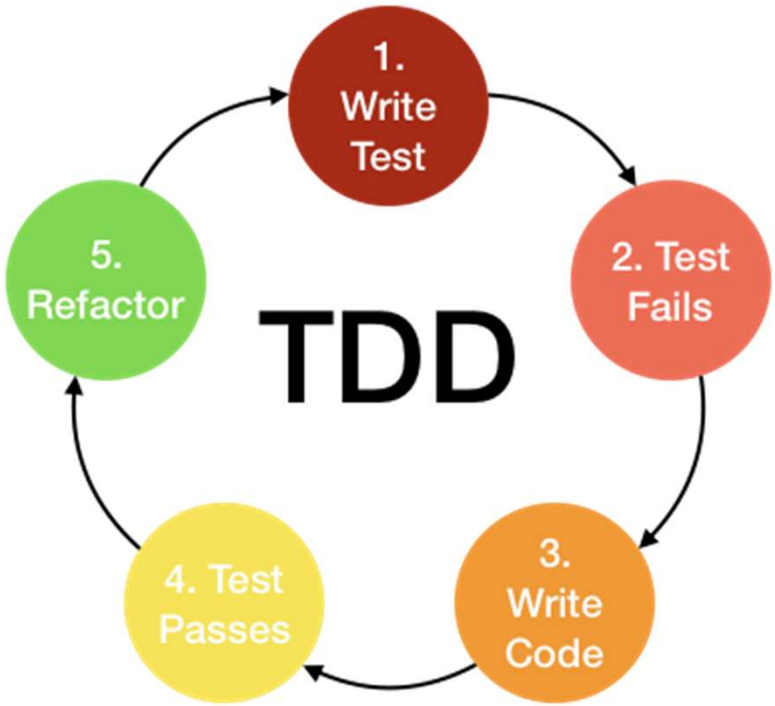
Ensure that any methods which returned a promise still return a promise in the mock

Create mocks for any complex values that are returned from methods which are required for tests

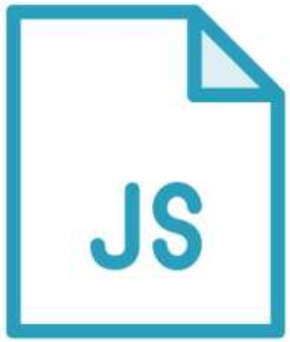
Test Driven Development



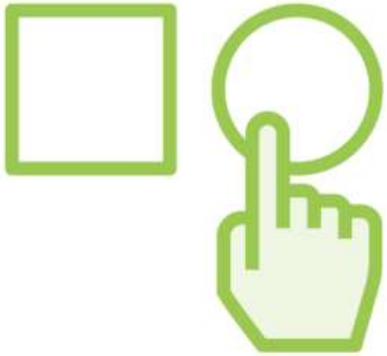
Test Driven Development



What is Jest?



A library installed via NPM or Yarn and run via the command line

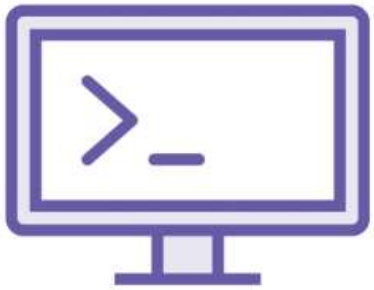


Similar to popular test-runners but with handy extra features

A tool made by a team including members of the React team

Jasmine / Mocha + Chai

Test-runner that organizes tests into “describe” and “it” blocks (or “suite” and “test”)

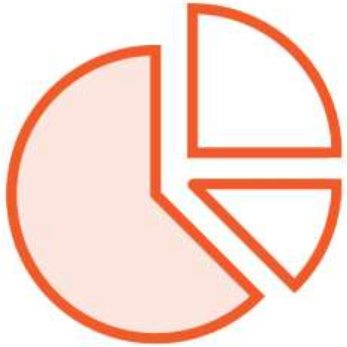


All assertions inside tests are verified whenever the test-runner is invoked, e.g., with command line

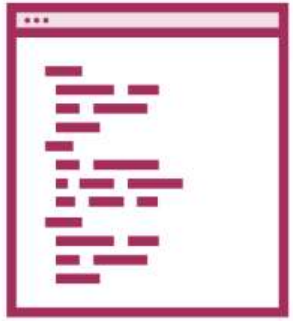
Doesn’t include module mocking or snapshots

Jest

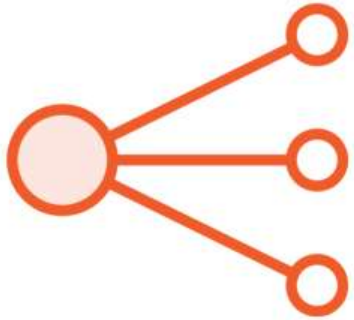
Built on top of Jasmine / Mocha + Chai



Adds snapshot testing, module mocking and many other useful features



Includes superior assertion library, CLI



Works with or without React

Jasmine v. Mocha v. Jest

	Mocha / Chai	Jasmine	Jest
Runs tests	Yes	Yes	Yes
Asynchronous	Yes	Yes	Yes
Spies included	No	Yes	Yes
Mocking / Module mocking	No	Yes / No	Yes
Snapshot testing	No	No	Yes

Jest & Jest CLI

JEST

The actual test-runner library which you use to execute your tests

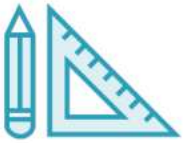
JEST CLI

A tool that you use from the command line to run and provide configuration options to Jest (the test runner)

Jest Versions

Latest version at time of October 2021: 27.2.5

Expect slightly different API/methods when entering a real-world project or moving from one workplace to another



Versions in 0.*.* and 1*.*.* families work differently than version in the 2*.*.* family (which we are using)

Practical Jest Usage 101

Watcher is triggered by NPM on development start script

CI suite runs tests and rejects failing PRs on integration server

Senior devs review coverage reports and CI hooks

End user is never aware of Jest, only that the application works

Different Jest Skillsets

Junior Developer

Proper use of “describe/it” or “suite/test” blocks

Understand which tests protect against regression and wrong functionality, and which are just “window dressing”

Practical strategies for resolving regression scenarios (not just disabling the test or updating the snapshot!)

Writing good, robust tests that properly utilize the existing API

Senior Developer

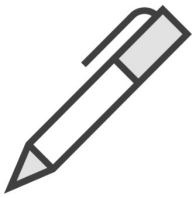
Configure Jest via CLI arguments and the configuration file

Package the correct Jest configurations into NPM scripts for usage by CI and junior developers

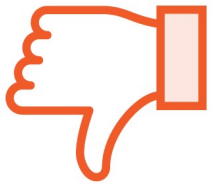
Analyze code coverage reports and test output in order to advise management

Writing tests that go beyond snapshots and truly verify the core functionality

Common Jest Pitfalls



Tests are not written

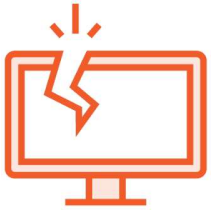


No integration with version control

Jest is not integrated into devs' workflow

Devs skip tests instead of fixing them

No integration with Deployment / CI



Tests do not protect against critical errors

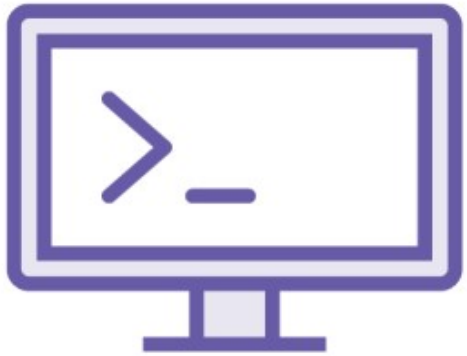
Jest Installation

Installed via NPM like many other libraries

Local installation should determine version, but in practice CLI may call local or global installation

CI installs Jest and CLI automatically, usually based on package.json

Running Tests



Tests are run by using the Jest CLI (typing “jest” followed by arguments in the command line)

test
test-watch
test-e2e
test-update
test-prod

The correct configuration for various different test patterns are stored as NPM scripts

In practice, tests are “run” by CI software and “watched” by everything else

Creating Test Files

__tests__/*.js

Any files inside a folder named __tests__ are considered tests

***.spec.js**
***.test.js**

Any files with .spec or .test in their filename are considered tests

What option to choose?

Tests in Their Own Folder

Easily distinguish between test and non-test files

Unrelated files can share a folder (i.e., a “loginService” test and a “cryptoHash” test)

Very easy to isolate a particular set of tests that are in the same folder

Tests can be named anything but must be inside an appropriately named folder (i.e., *__tests__*), to be recognized

Tests Alongside Components

Which files are components and which files are spec is not as obvious

Tests are always adjacent to the files they apply to

Unrelated tests are less likely to share a folder

Tests must have the correct naming pattern to be recognized, i.e., **.test.js*

Possible to isolate tests based on name patterns, i.e., *user-**

Watching Tests

In “watch mode”, tests are run automatically as files change

Only tests pertaining to changed files are run

Jest detects changes automatically

Actively prevents regression

Jest Globals: Describe and It

DESCRIBE
(SUITE)

An optional method for grouping any number of *it* or *test* statements

IT
(TEST)

Method which you pass a function to, that function is executed as block of tests by the test runner

Gherkin Syntax

DSL (Domain-Specific Languaje)
for business behavior descriptions

remove logic details from behavior tests.

https://cucumber.io/docs/gherkin

Given,
When,
Then

(And, But)

```
describe('given the function strictEquals', () => {
  describe('when function receives 1 and 1', () => {
    test('then it returns true', () => {
    });
  });
});
```

Test definition pattern AAA


```
describe('given the function st
  describe('when function recei
    test('then it returns true'
       ------------------->
    });
  });
});
```

```
//Arrange
const data1 = 1;
const data2 = 1;

//Act
const result = strictEquals(data1, data2);

//Assert
expect(result).toBeTruthy();
```

Expectativa

- En las funciones it se utiliza el método expect("expresion") que define las expectativas de la prueba sobre una determinada expresión o variable
- Una expectativa comienza con la función expect().
- Toma un valor y llama al cotejo (matcher) para comprobar si la expectativa se ha cumplido.

```
expect(variable).toEqual(valorEsperado)
```

al resultado puede concatenársele la definición de la evaluación a realizar gracias al conjunto de métodos soportados en el *framework* Jasmine

Cotejos (Mathchers)

Igualdad	expect(mixed).toEqual(mixed);
Negación	expect(mixed).not.to…();
Identidad	expect(mixed).toBe(mixed); expect(mixed).toStrictEqual(mixed)
Instancia	expect(mixed).toBeInstanceOf(Class)
Si o No	expect(mixed).toBeTruthy(); expect(mixed).toBeFalsy();

Comprobar si un elemento está presente	expect(array).toContain(member);
	expect(string).toContain(string);
	expect(item).toContainEqual(item)

Comprobar si un elemento está definido	expect(mixed).toBeDefined(); expect(mixed).toBeUndefined();
Nulidad	expect(mixed).toBeNull();
Comprobar si un elemento es NaN	expect(number).toBeNaN();

Comparaciones	expect(number| bigint).toBeGreaterThan(number| bigint);
	expect(number| bigint).toBeGreaterThanOrEqual(number | bigint)
	expect(number| bigint).toBeLessThan(number| bigint);
	expect(number| bigint).toBeLessThanOrEqual(number | bigint)

Proximidad	expect(number).toBeCloseTo(number, decimalPlaces);

Cotejos (Mathchers)

Comparaciones con Regular Expressions	expect(mixed).toMatch(regexp | string);
	.toMatchObject(object) .toMatchSnapshot(propertyMatchers?, hint?) .toMatchInlineSnapshot(propertyMatchers?, inlineSnapshot)
Errores	.toThrow(error?) .toThrowErrorMatchingSnapshot(hint?) .toThrowErrorMatchingInlineSnapshot(inlineSnapshot)
spy matchers	.toHaveBeenCalled() .toHaveBeenCalledTimes(number) .toHaveBeenCalledWith(arg1, arg2, ...) .toHaveBeenLastCalledWith(arg1, arg2, ...) .toHaveBeenNthCalledWith(nthCall, arg1, arg2, ...) .toHaveReturned() .toHaveReturnedTimes(number) .toHaveReturnedWith(value) .toHaveLastReturnedWith(value) .toHaveNthReturnedWith(nthCall, value) .toHaveLength(number) .toHaveProperty(keyPath, value?)

Jest Globals: BeforeEach and BeforeAll

BeforeEach runs a block of code before each test

Useful for setting up databases, mock instances, etc.

BeforeAll runs code just once, before the first test

Jest Globals: AfterEach and AfterAll

Inverse versions of BeforeEach and BeforeAll

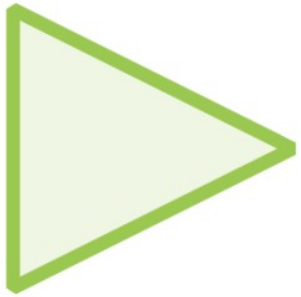
Runs a block of code after each test (or after the last test)

Useful for closing open connections, terminating sub-processes

Skipping and Isolating Tests

Skipping a test results in that test not being run

Mark a test as skip



Isolating a test results in only it (and any other isolated tests) running

Mark a test as only

What Are Asynchronous Tests?

Contains assertions (like a regular test)

Does not complete instantaneously

Can take varying amounts of time, even an unknown amount of time

Jest must be notified that test is complete

Defining Asynchronous Tests

Invoke the done() callback that is passed to the test

Return a promise from a test

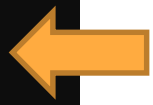
Pass an async function to describe

Coding Asynchronous Tests

```
it('async test 1', (done) => {
  setTimeout(done, 100);
});

it('async test 2', () => {
  return new Promise((resolve) =>
     setTimeout(resolve, 100));
});

it('async test 3', async () =>
await delay(100));
```



The ways of formatting an async test shown here are roughly equivalent

Delay is a method that returns a promise

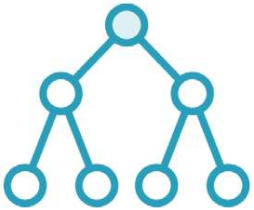
Mock Functions

Also known as “spies”

No side-effects

Counts function calls

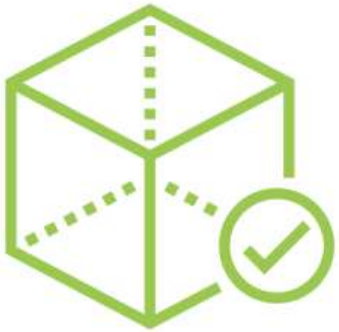
Records arguments passed when called



Can be “loaded” with return values

Return value must approximate original

Creating Mock Files for modules



Appropriately named NPM mocks are loaded automatically

Mocks must reside in a *__mocks__* folder next to mocked module

NPM modules and local modules can both be mocked