

IN JEST

An Overview and Review of an Open Source Testing Framework for Javascript

For CPSC 542

By Jason Ebueng

IN JEST

This presentation will cover Jest, an open-source testing tool from Facebook.

There will be a brief history about testing.

Then there will be a demonstration with Jest.

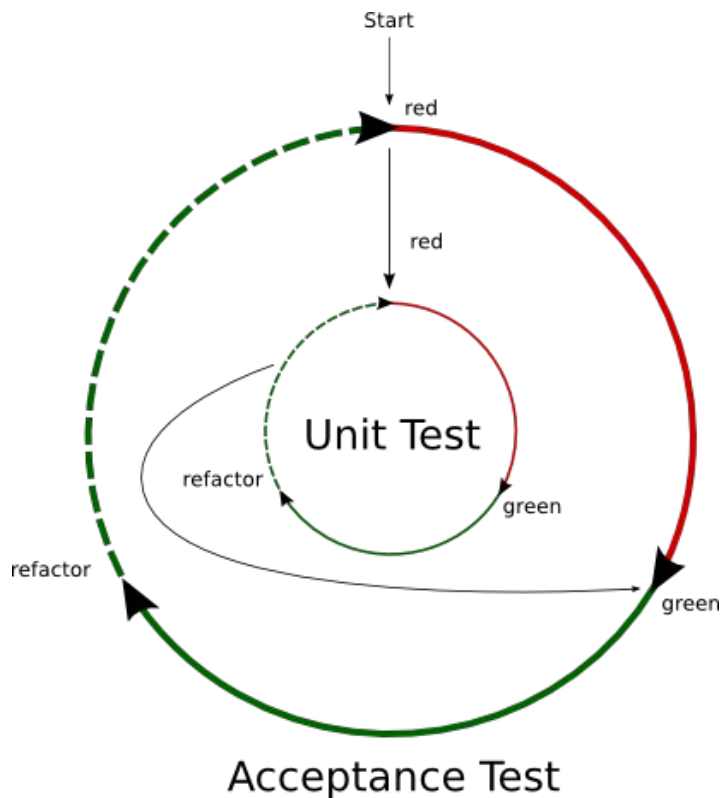
This analysis should provide insight on the current trends and testing tools for front-end development.

THE UNIT TEST

Building the entire application has all these features implemented with code, so thus *unit testing* is the means of testing those individual components of the system.

Instagram can be broken down into features - there are *comments*, *friends*, and of course, *pictures*.

By testing the behavior of those individually, integrating them into a system should be much easier.

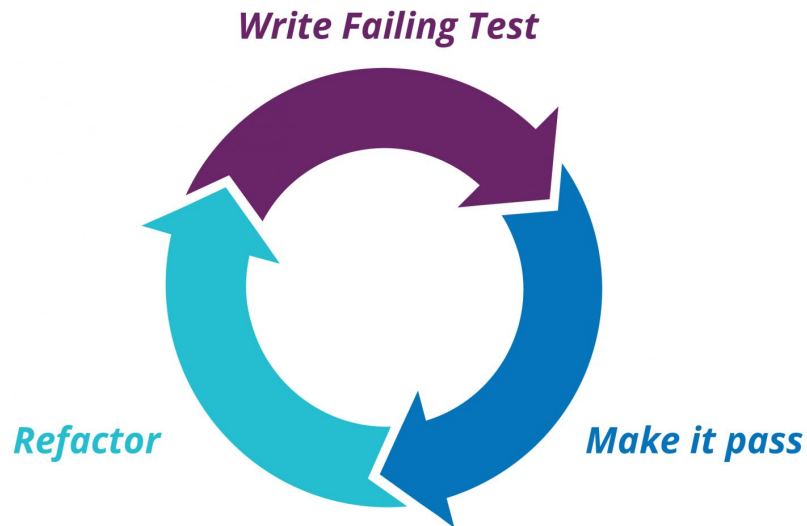


THE UNIT TEST

In the 1940s, it said that ENIAC programmers applied tests with artillery software.

In 1957, *Program Checkout* documented the approach of unit testing in one of the earliest examples of testing.

There saw a resurgence in the 2000s with Software Engineer Kent Beck.



FROM RSPEC TO JEST



FROM RSPEC TO JEST

Usage [\[edit\]](#)

Describing the behavior of objects [\[edit\]](#)

As mentioned above, RSpec provides a domain-specific language to describe the behavior of [objects](#). The [keywords](#) used in RSpec are similar to the ones used in other languages and/or TDD frameworks.^[6] For example, if the keywords used in Test::Unit are considered, they can be mapped to the RSpec keywords as follows:

- Assertion becomes *expectation*
- Test method becomes *Example code*
- Test case becomes *Example group*

There are many such keywords which are used in the same context but with the similar names. The syntax of RSpec provides the ease of readability and describes the behavior of the code thereby providing freedom to the programmer. Every testing framework works in the following flow - given some context, when some event occurs, what outcome is expected. The methods like `describe()`, `context()` and `it()` form the analogy and the skeleton respectively of the test code.

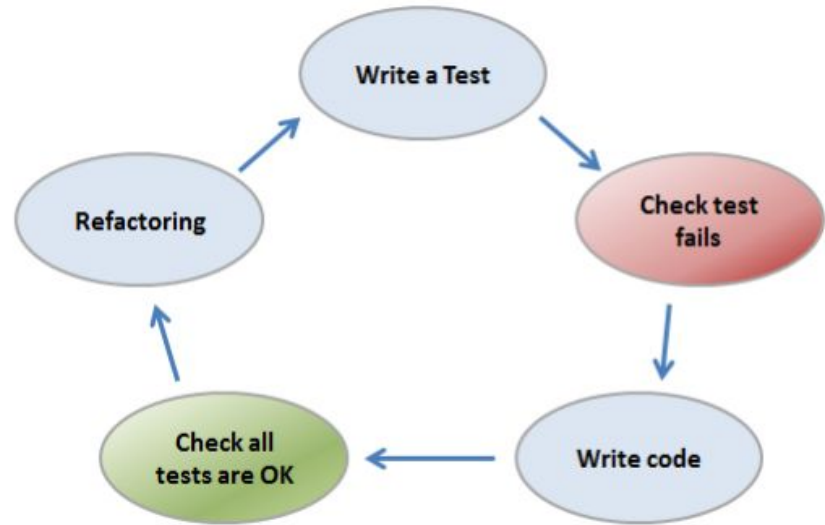
```
function sum(a, b) {  
  return a + b;  
}  
module.exports = sum;
```

Then, create a file named `sum.test.js`. This will contain our actual test:

```
const sum = require('./sum');  
  
test('adds 1 + 2 to equal 3', () => {  
  expect(sum(1, 2)).toBe(3);  
});
```

PRINCIPLES

- 1.) Add Tests
- 2.) Run Tests
- 3.) Write Code
- 4.) Run Tests
- 5.) Refactor Code



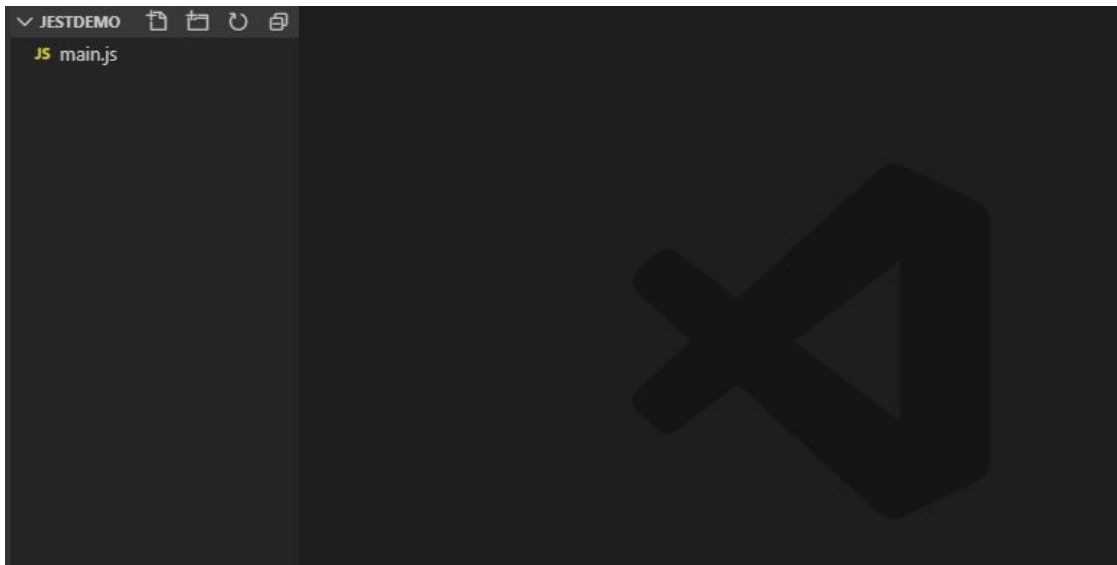
<https://hackernoon.com/introduction-to-test-driven-development-tdd-61a13bc92d9>

JEST SETUP

A Windows 10 Environment with the latest JavaScript environment setup.

In particular, the “Windows On Linux” feature will be utilized.

Furthermore, code will be demonstrated with the Visual Code IDE.



JEST SETUP

NPM is a package manager for JavaScript, and an essential tool for front-end development.

Initializing an NPM project will create a package.json file, which sets up the rest of the project.

By setting commands in the scripts section, we can enable the Jest tool.

```
() package.json > {} devDependencies
1  {
2    "name": "jestdemo",
3    "version": "1.0.0",
4    "description": "",
5    "main": "main.js",
6    "scripts": {
7      "test": "jest"
8    },
9    "author": "",
10   "license": "ISC",
11   "devDependencies": {
12     "jest": "^24.9.0"
13   }
14 }
15
```

1.) WRITE THE TEST

Before writing code, tests and expectations are planned.



```
{ } package.json JS sum.test.js X
1  const sum = require('./sum');
2
3  test('adds 1 + 2 to equal 3', () => {
4    expect(sum(1, 2)).toBe(3);
5  });
6
7  test('adds 2 + 2 to equal 5', () => {
8    expect(sum(2, 2)).toBe(4);
9  });
10
11 |
```

2.) RUN TESTS

Still before writing any code, the test is ran first. Failure is anticipated.

This establishes any more expectations and helps targets any external issues.

```
jason@DESKTOP-B4P5PSG:/mnt/c/Users/Jason/projects/jestdemo$ npm run test
> jestdemo@1.0.0 test /mnt/c/Users/Jason/projects/jestdemo
> jest

FAIL ./sum.test.js
  ● Test suite failed to run

    ReferenceError: sum is not defined

       1 | module.exports = sum;
         | ^

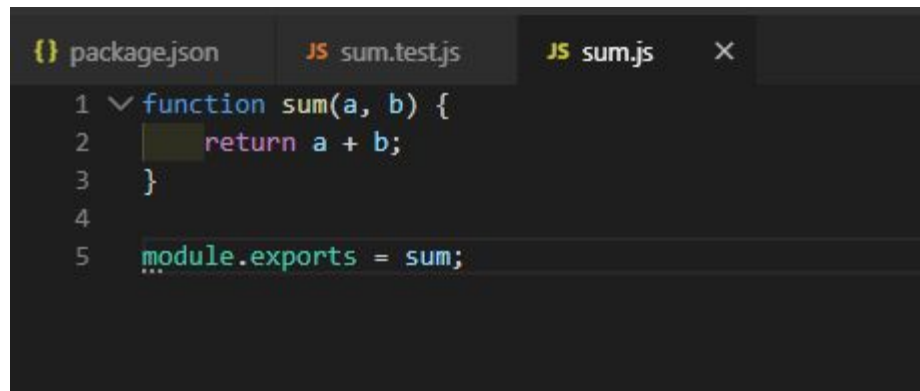
    at Object.<anonymous> (sum.js:1:1)
    at Object.<anonymous> (sum.test.js:1:1)

Test Suites: 1 failed, 1 total
Tests:      0 total
Snapshots: 0 total
Time:      40.693s
```

3.) WRITE CODE

The code is now finally written against expectations.

It should return the expected output from the tests.



```
{ } package.json JS sum.test.js JS sum.js X
1  function sum(a, b) {
2    return a + b;
3  }
4
5  module.exports = sum;
```

4.) RUN TESTS

Now that code has been implemented, running the test now will yield different results.

```
jason@DESKTOP-B4P5PSG:/mnt/c/Users/Jason/projects/jestdemo$ npm run test

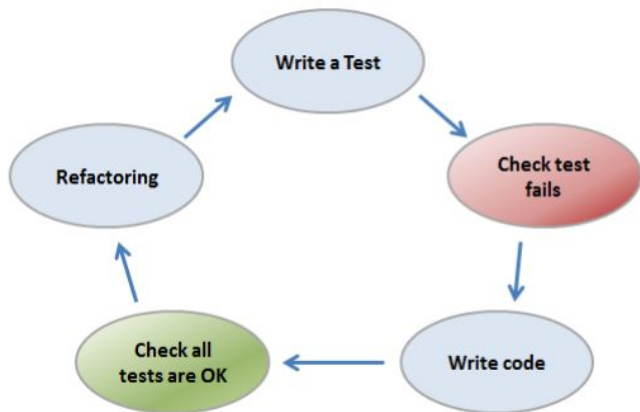
> jestdemo@1.0.0 test /mnt/c/Users/Jason/projects/jestdemo
> jest

PASS ./sum.test.js
  ✓ adds 1 + 2 to equal 3 (5ms)
  ✓ adds 2 + 2 to equal 5 (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        27.999s
Ran all test suites.
```

5.) REFACTOR

Pass or fail, the developer can refine or fix their code until the results are accepted.



```
{ } package.json JS sum.test.js JS sum.js X
1  var sum = (a, b) => a + b;
2
3  module.exports = sum;
```

```
jason@DESKTOP-B4P5PSG:/mnt/c/Users/Jason/projects/jestdemo$ npm run test

> jestdemo@1.0.0 test /mnt/c/Users/Jason/projects/jestdemo
> jest

PASS ./sum.test.js
  ✓ adds 1 + 2 to equal 3 (6ms)
  ✓ adds 2 + 2 to equal 5 (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        6.016s
Ran all test suites.
```

REPEATING THE PROCESS

Now that the code has passed the tests, the developer can safely progress towards other goals.

Suppose the developer wanted to improve functionality significantly to code.

The sum function only accepts 2 arguments - what if we wanted to accept more?

```
package.json  JS sum.test.js  JS sum.js  X
1  var sum = (a, b) => a + b;
2
3  module.exports = sum;
```

```
var sum = (...args) => [...args].reduce((a, b) => a + b, 0)

module.exports = sum;
```

```
Time: 6.016s
Ran all test suites.
jason@DESKTOP-B4P5PSG:/mnt/c/Users/Jason/projects/jestdemo$ npm run test

> jestdemo@1.0.0 test /mnt/c/Users/Jason/projects/jestdemo
> jest

PASS ./sum.test.js (11.143s)
  ✓ adds 1 + 2 to equal 3 (23ms)
  ✓ adds 2 + 2 to equal 5 (1ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 43.522s
Ran all test suites.
```

REFINEMENT

Tests are reliably way of documenting code.

However, tests are only good as their developers can make them.

Being specific and creative helps target unique cases to the test.

For example, now that we have implemented a multiple arguments for the sum function, there should be a check for massive lists of sums.

```
1  const sum = require('./sum');
2
3  ✓ test('adds 1 + 2 to equal 3', () => {
4    expect(sum(1, 2)).toBe(3);
5  });
6
7  ✓ test('adds 2 + 2 to equal 5', () => {
8    expect(sum(2, 2)).toBe(4);
9  });
10
11 ✓ test('adds 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 to equal 10', () => {
12   expect(sum(1,1,1,1,1,1,1,1,1,1)).toBe(10);
13 });
14
15 |
```


CAVEATS

Due to being a testing tool for front-end development, there are several nuances to consider.

For example, JavaScript isn't strongly typed and doesn't enforce data types. This allows rapid development, but leaves much ambiguity and bugs in larger projects.

JavaScript now has TypeScript and Flow, which are layered to enforce data types.

Another caveat is that front-end development code is asynchronous. This brings unique issues both internally of JavaScript and externally.

Developers doesn't have direct access to a database, and ultimately rely on back-end services.

ADDITIONAL CONCEPTS

Continuous Integration is constantly testing updated code and integrating the code into a live system.

Testing tools such as Jest are supported by platforms such as CircleCI and Github.

Github hosts the code changes, then enables deployment tools to run tests.

Integrate seamlessly

Whether you're pushing code to the cloud or on-premise, CircleCI easily integrates with GitHub and GitHub Enterprise.



 RUNNING

 SUCCESS

Start building now

Setup your project in minutes. CircleCI mirrors your GitHub team permissions and privileges, which means there are no plugins to install or credentials to create.

ADDITIONAL CONCEPTS

Mocks are test data that integrate better for the code.

These can be simple text or key-value data objects,

However testing the data directly as the data is manipulated by the environment is important to distinguish.

A good case to mock are API calls, database requests, and manipulated data.

```
1  test("returns undefined by default", () => {  
2    const mock = jest.fn();  
3  
4    let result = mock("foo");  
5  
6    expect(result).toBeUndefined();  
7    expect(mock).toHaveBeenCalled();  
8    expect(mock).toHaveBeenCalledTimes(1);  
9    expect(mock).toHaveBeenCalledWith("foo");  
10  });
```

mock_basic.js hosted with ❤ by GitHub

ADDITIONAL CONCEPTS

One powerful feature of Jest is snapshot testing.

React is a performant front-end tool for JavaScript, enabling HTML/CSS to work seamlessly with JS to make single-page applications.

It relies on virtual DOM technology, which helps handle renders.

Jest can directly inspect these as snapshots.

```
import React from 'react';
import Link from '../Link.react';
import renderer from 'react-test-renderer';

it('renders correctly', () => {
  const tree = renderer
    .create(<Link page="http://www.facebook.com">Facebook</Link>)
    .toJSON();
  expect(tree).toMatchSnapshot();
});
```

The first time this test is run, Jest creates a [snapshot file](#) that looks like this:

```
exports[`renders correctly 1`] = `
<a
  className="normal"
  href="http://www.facebook.com"
  onMouseEnter={[Function]}
  onMouseLeave={[Function]}
>
  Facebook
</a>
`;
```

<https://jestjs.io/docs/en/snapshot-testing>

ADDITIONAL CONCEPTS

Because data can come in all shapes, sizes, and types, it's important to clarify what attributes are being tested.

“Matchers” are a fundamental concept between testing tools.

Matcher functions help compare and evaluate data.

`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);  
    }  
  }  
});
```

<https://jestjs.io/docs/en/using-matchers>

ADDITIONAL CONCEPTS

Matchers concern itself with “Truthiness” of data, which can be abstract.

The existence of data, even at the binary level, has three states - true, false, and null.

Null is the lack of presence or existence of data, but JavaScript’s approach to truthiness isn’t consistent with other languages.

Truthiness

In tests you sometimes need to distinguish between `undefined`, `null`, and `false`, but you sometimes do not want to treat these differently. Jest contains helpers that let you be explicit about what you want.

- `toBeNull` matches only `null`
- `toBeUndefined` matches only `undefined`
- `toBeDefined` is the opposite of `toBeUndefined`
- `toBeTruthy` matches anything that an `if` statement treats as true
- `toBeFalsy` matches anything that an `if` statement treats as false

```
test('null', () => {  
  const n = null;  
  expect(n).toBeNull();  
  expect(n).toBeDefined();  
  expect(n).not.toBeUndefined();  
  expect(n).not.toBeTruthy();  
  expect(n).toBeFalsy();  
});  
  
test('zero', () => {  
  const z = 0;  
  expect(z).nottoBeNull();  
  expect(z).toBeDefined();  
  expect(z).not.toBeUndefined();  
  expect(z).not.toBeTruthy();  
  expect(z).toBeFalsy();  
});
```

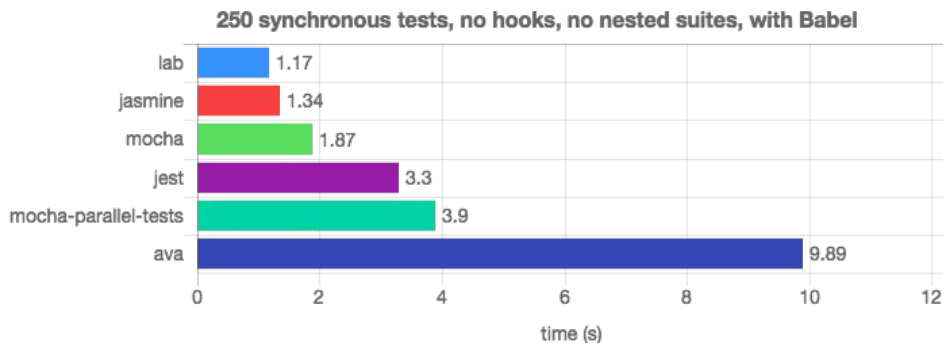
<https://jestjs.io/docs/en/using-matchers>

LIMITATIONS OF JEST

Other JavaScript testing tools, such as Mocha and Jasmine, offer similar TDD approaches but differ in architecture and contemporary practices.

Since Jest was created with React, they both promote a “functional programming” style, whereas Mocha is a more traditional tool that is more flexible and less opinionated on how to test.

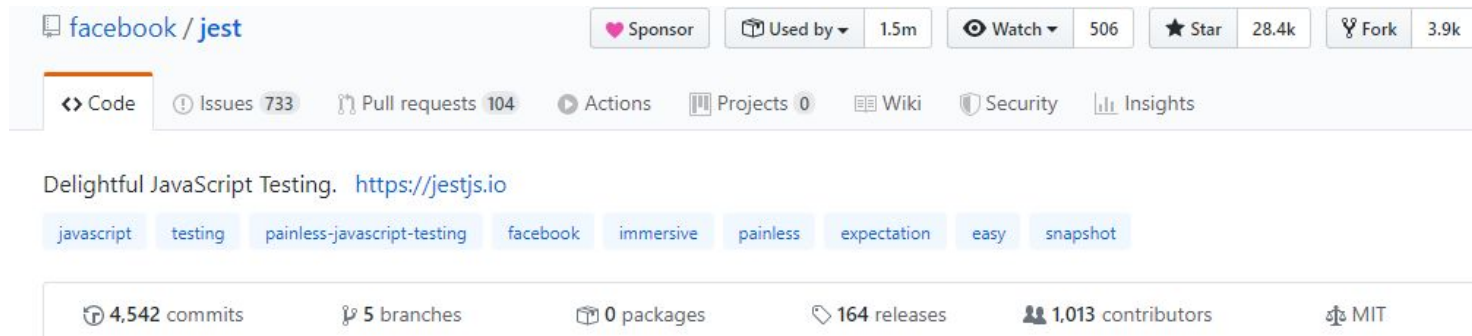
Jest’s biggest weakness is relatively young age - older frameworks have more options and tools supporting it.



<https://medium.com/dailyjs/javascript-test-runners-benchmark-3a78d4117b4>

BENEFITS OF JEST

The benefits of Jest also includes its relatively young age, because it's more suited for modern practices and technologies. Due to its popularity and adoption by developers, updates and maintenance see significant activity.



The screenshot displays the GitHub repository for Jest, maintained by Facebook. At the top, the repository name 'facebook / jest' is shown. To the right, there are several interactive buttons: 'Sponsor' (with a heart icon), 'Used by' (with a dropdown arrow and '1.5m' users), 'Watch' (with a dropdown arrow and '506' watchers), 'Star' (with a star icon and '28.4k' stars), and 'Fork' (with a fork icon and '3.9k' forks). Below these, a navigation bar includes links for 'Code', 'Issues' (733), 'Pull requests' (104), 'Actions', 'Projects' (0), 'Wiki', 'Security', and 'Insights'. The main description reads 'Delightful JavaScript Testing.' followed by the website 'https://jestjs.io'. Below the description are several topic tags: 'javascript', 'testing', 'painless-javascript-testing', 'facebook', 'immersive', 'painless', 'expectation', 'easy', and 'snapshot'. At the bottom, a summary bar shows repository statistics: '4,542 commits', '5 branches', '0 packages', '164 releases', '1,013 contributors', and the license 'MIT'.

facebook / jest

Sponsor Used by 1.5m Watch 506 Star 28.4k Fork 3.9k

Code Issues 733 Pull requests 104 Actions Projects 0 Wiki Security Insights

Delightful JavaScript Testing. <https://jestjs.io>

javascript testing painless-javascript-testing facebook immersive painless expectation easy snapshot

4,542 commits 5 branches 0 packages 164 releases 1,013 contributors MIT

BENEFITS OF JEST

The benefits of Jest also includes its relatively young age, because it's more suited for modern practices and technologies. Due to its popularity and adoption by developers, updates and maintenance see significant activity.

Architecture.md	Add Jest Architecture overview to docs. (#7449)	last year
BypassingModuleMocks.md	Update BypassingModuleMocks.md (#8470)	6 months ago
CLIMd	docs: Add collectCoverage to Reference (#8996)	last month
Configuration.md	feat: allow reporters to be default exports (#9161)	7 days ago
DynamoDB.md	Add DynamoDB usage example (#8319)	7 months ago
Es6ClassMocks.md	Docs: More inclusive wording (#9045)	last month
ExpectAPIMd	Clarify usage of isNot in docs (#9123)	11 days ago
GettingStarted.md	Update link to webpack docs (#9078)	23 days ago
GlobalAPIMd	chore: bump prettier (#9153)	9 days ago
JestCommunity.md	upgrade prettier (#6344)	2 years ago
JestObjectAPIMd	docs: Add 'setupFilesAfterEnv' and 'jest.setTimeout' example	2 months ago
JestPlatform.md	Fix typo in jest-docblock description (#7635)	10 months ago
ManualMocks.md	Docs: Removed condescending language (#9040)	last month
MigrationGuide.md	Docs: Removed condescending language (#9040)	last month
MockFunctionAPIMd	chore: bump prettier (#9153)	9 days ago
MockFunctions.md	fix mockReturnValueOnce code example (#8480)	7 days ago
MongoDB.md	Simplify mongodb example by using preset (#8318)	7 months ago
MoreResources.md	Docs: Removed condescending language (#9040)	last month
Puppeteer.md	docs: Add notice for known bug with puppeteer (#8527)	6 months ago
SetupAndTeardown.md	Docs: Removed condescending language (#9040)	last month
SnapshotTesting.md	Docs: Removed condescending language (#9040)	last month
TestingAsyncCode.md	Docs: Removed condescending language (#9040)	last month
TestingFrameworks.md	add Chinese Jest work with AngularJS tutorial (#8828)	3 months ago
TimerMocks.md	docs: s/Times/Time's (#8440)	6 months ago

BENEFITS OF JEST

Jest also isn't strictly for front-end technologies, nor does it care about React. Other frameworks such as NextJS and Angular can utilize it as well.

angular	chore: add Angular example (#8905)
async	chore(breaking): remove regenerator-runtime injection (#7595)
automatic-mocks	chore(breaking): remove regenerator-runtime injection (#7595)
enzyme	chore: use property initializer syntax (#8117)
getting-started	chore(breaking): remove regenerator-runtime injection (#7595)
jquery	chore(breaking): remove regenerator-runtime injection (#7595)
manual-mocks	chore(breaking): remove regenerator-runtime injection (#7595)
module-mock	chore(breaking): remove regenerator-runtime injection (#7595)
react-native	[ImgBot] Optimize images (#9092)
react-testing-library	docs: add comment about 'cleanup' in react-testing-library exam... (#9144)
react	refactor: Replaces findDOMNode() with refs (#9124)
snapshot	chore: use property initializer syntax (#8117)
timer	feat: add fake timers implementation backed by Lolex (#8897)
typescript	chore: get rid of custom abstraction on top of execa in e2e tests (#8901)

BENEFITS OF JEST

Jest, when combined with version control and deployment tools such as Github and CircleCI, enables rapid delivery and promotes Agile development.



✓ Verified by GitHub
GitHub confirms that this app meets the [requirements for verification](#).

Categories

Continuous integration

Mobile CI

Application

Travis CI

Set up a free trial

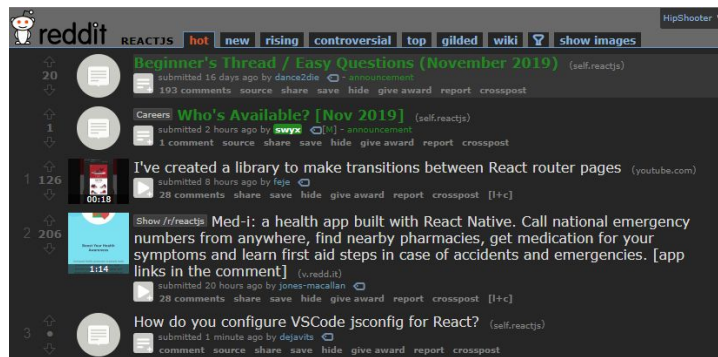
Travis CI enables your team to test and ship your apps with confidence. It's built for everyone and for projects and teams of all sizes, supporting over 20 different languages out of the box, including Javascript and Node.js, Ruby, PHP, Python, Mac/iOS, as well as Docker, while giving you full control over the build environment to customize it to your own needs.

Travis CI is trusted by hundreds of thousands of open source projects, teams, and developers.

DEVELOPER INSIGHT

What I appreciate the most about Jest is it's popularity. It's been highlighted as a benefit, but it's value is tremendous to me as a web developer in 2019.

Not only can I find communities in StackOverFlow, I can go through forums such as Reddit or chat channels in Discord or Slack to find a healthy tech community.



Reactiflux

WELCOME TO

We're a chat community of 80,000+ React JS , React Native , Redux , Jest , Relay , and GraphQL  developers. We hold Q&A's with Facebook Engineers  and other developers  in the community . Come chat about tech related to React & JavaScript or ask for help!

Join Reactiflux

Q&A Schedule

DEVELOPER INSIGHT

Another benefit I appreciate is the looser enforcement of syntax.

Unlike Python, there's an approach of embracing synonyms and multiple names for the same functionality. This is rather contentious, but I believe it helps unite more developers.

For example, `it()` and `test()` are the same but developers from different frameworks can use both in Jest.

```
environment.global.test = environment.global.it;  
environment.global.it.only = environment.global.fit;  
environment.global.it.skip = environment.global.xit;  
environment.global.xtest = environment.global.xit;  
environment.global.describe.skip = environment.global.xdescribe;  
environment.global.describe.only = environment.global.fdescribe;
```

CONCLUSION

Ultimately, while I enjoy Jest, it seems that the front-end development scene continues to change and evolve.

Functional programming is finally becoming more tangible with tools like Jest, but there are constantly new approaches being discovered as programming demands it.

Testing has a significant role in securing software. I think Jest will persist longer than other tools due to its modern features and community.

