



Jest: Frontend Testing done right

Holger Grosse-Plankermann



Who am I?

Developer/Consultant/Whatever

Taming the Web since the 2000

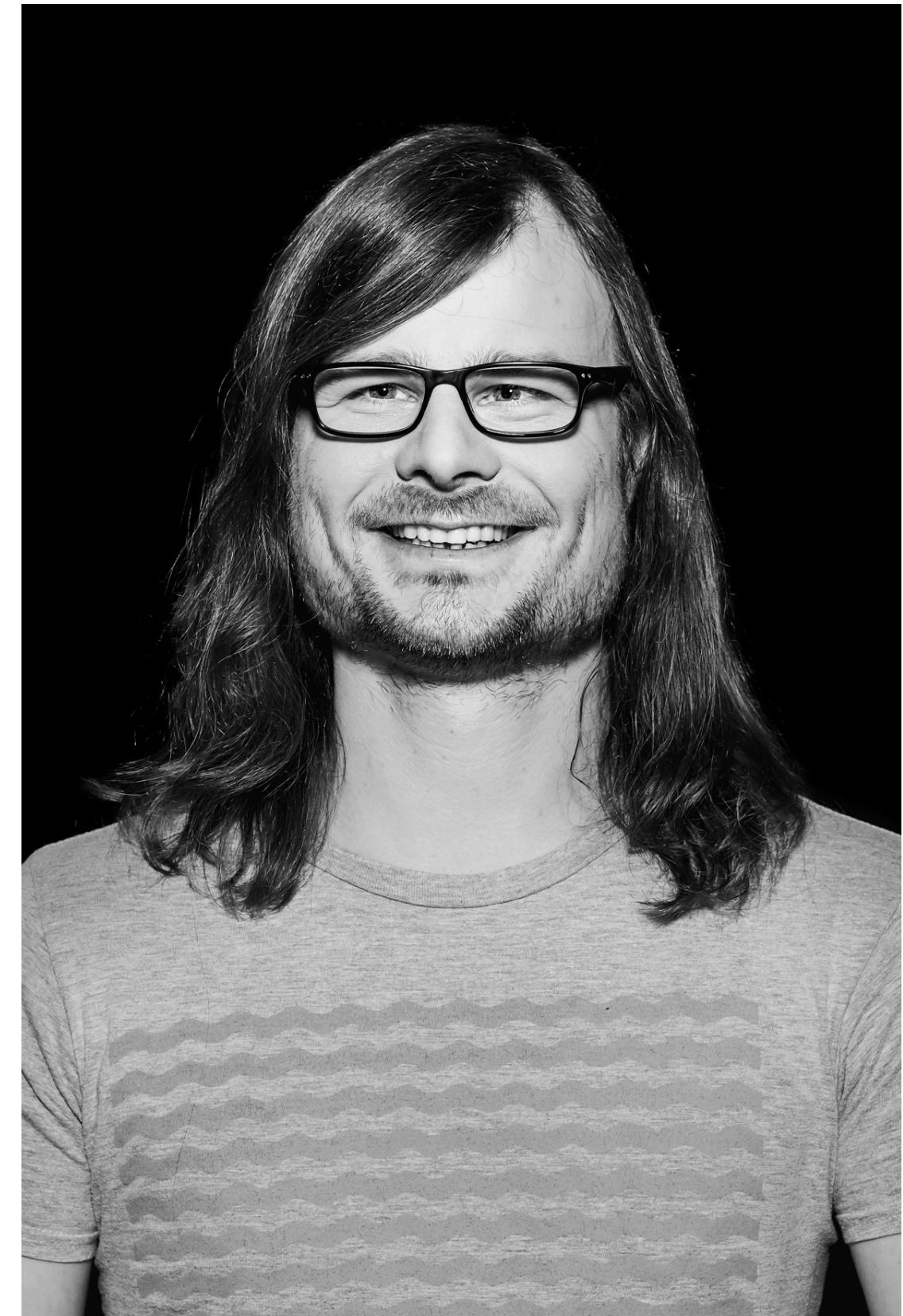
Compiled Mozilla for bonus features

Backend vs. Frontend dev

Podcaster <http://autoweird.fm>

@holgergp

<http://github.com/holgergp>



Holger Grosse-Plankermann

Consulting and so much more

International Events

Meetups
+1 Time

Trainings

Conferences

Knowledge Sharing

Blog
Video Production

Webinars

Social Media

Writing
Knowledge Leadership

Drone Project

distance
sensors

@Autoweird
fm

Podcasting

Public Speaking

Robotics

Show of hands

Do you test your frontend code?

-
- Unit Testing
 - Selenium
 - Macros

Show of hands

What Test framework are you using?

- Jest
- Mocha
- Karma
- Jasmine
- Something else?

AGENDA

Where are we heading?

- ⚡ Unit Tests in JavaScript
Where do I come from?
- → What is Jest
Why another library?
- ✅ Top 3 reasons for Jest
Number 3 will amaze you :)

(Unit-) Testing JS

Where do I come from



Early Days

- JavaScript === alert("Foo")
- Simple Formish Web Apps
- Seemingly no need for thorough testing like in the backend
- „Selenium is enough“

JQuery explosion

- Everybody wanted to build the new Myspace
 - ... with jQuery
- But frontend skills were still lacking
- *Let's do complicated backend stuff and finish the frontend with some sprinkles of jQuery ...*
 - ... And some more
 - jQuery Jenga
- Too „complicated“ to properly test our frontend code



A black and white photograph of a young man with dark hair, seen from the side and back. He is wearing a dark jacket over a light-colored shirt. He is walking away from the viewer through a hazy, possibly rainy or smoky, environment. The background is out of focus.

Rise of SPAs

- In the age of SPAs the need for proper testing on the client arose
 - Alongside the explosion of the ecosystem
- Many different approaches (Hey it's Javascript!!)
 - Jasmine
 - Mocha
 - QUnit
 -
- But: Terribly to set up!

Where are we now?

- SPAs/JS somehow mainstream
 - Often more code in the frontend than in the backend
- JS testing however is not
 - Console.log is still a thing





Why is that?

- Backend Java testing is well established
 - And Tooling is quite nice tbh
 - mvn clean install ftw
- JavaScript Testing used to be a mess
 - Tons of config and dependencies needed to be orchestrated
 - And that rumor is still in our heads

Testing in the frontend is important

- More and more complex code than in the backend
 - Rapid feedback
 - Safety net
 - Design!
 - Documentation
-
- Testing is a task many devs don't like
 - Don't let the tools stand in the way!
 - Help is on the way!



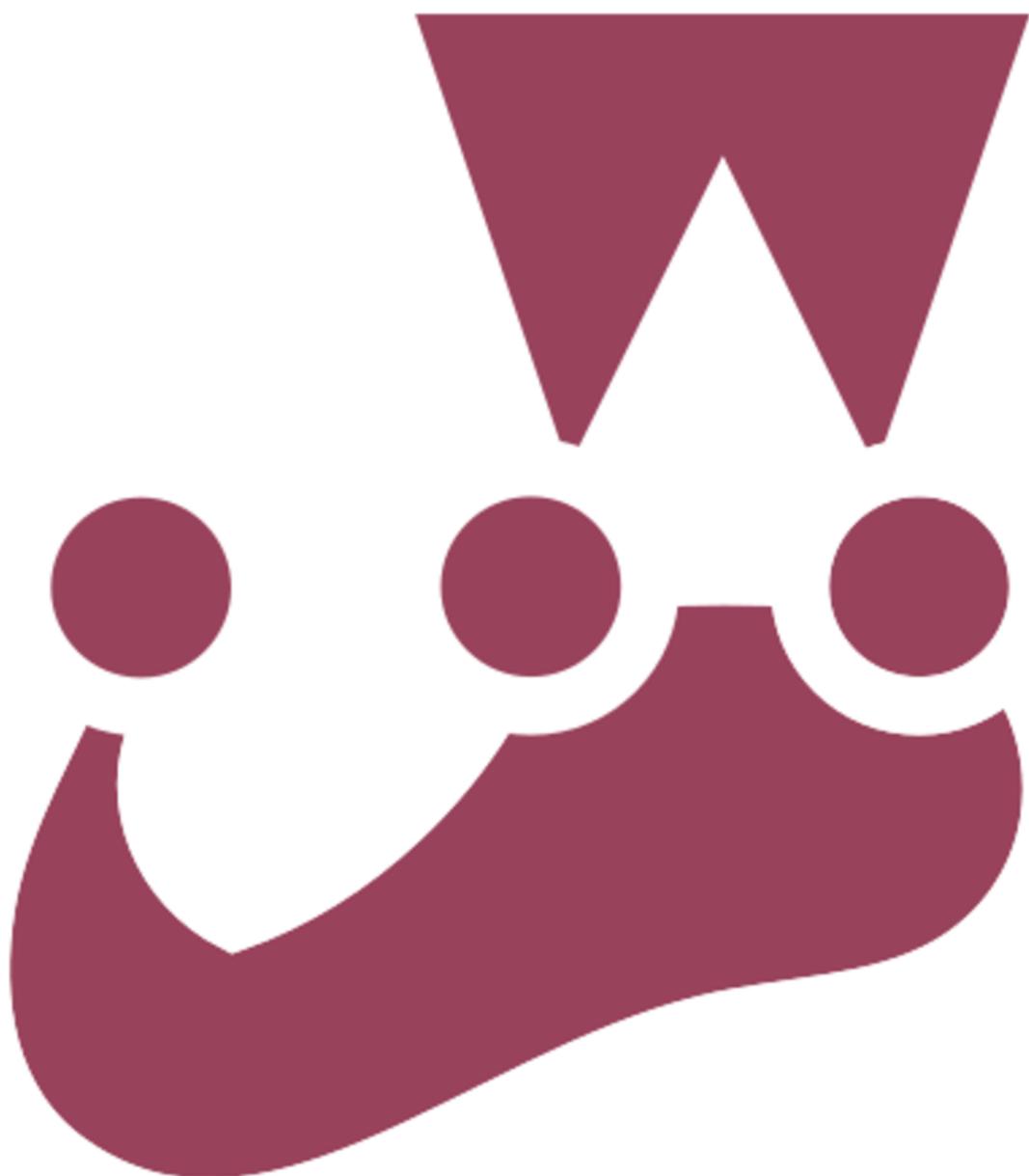
Enter Jest

A library that puts the fun in testing



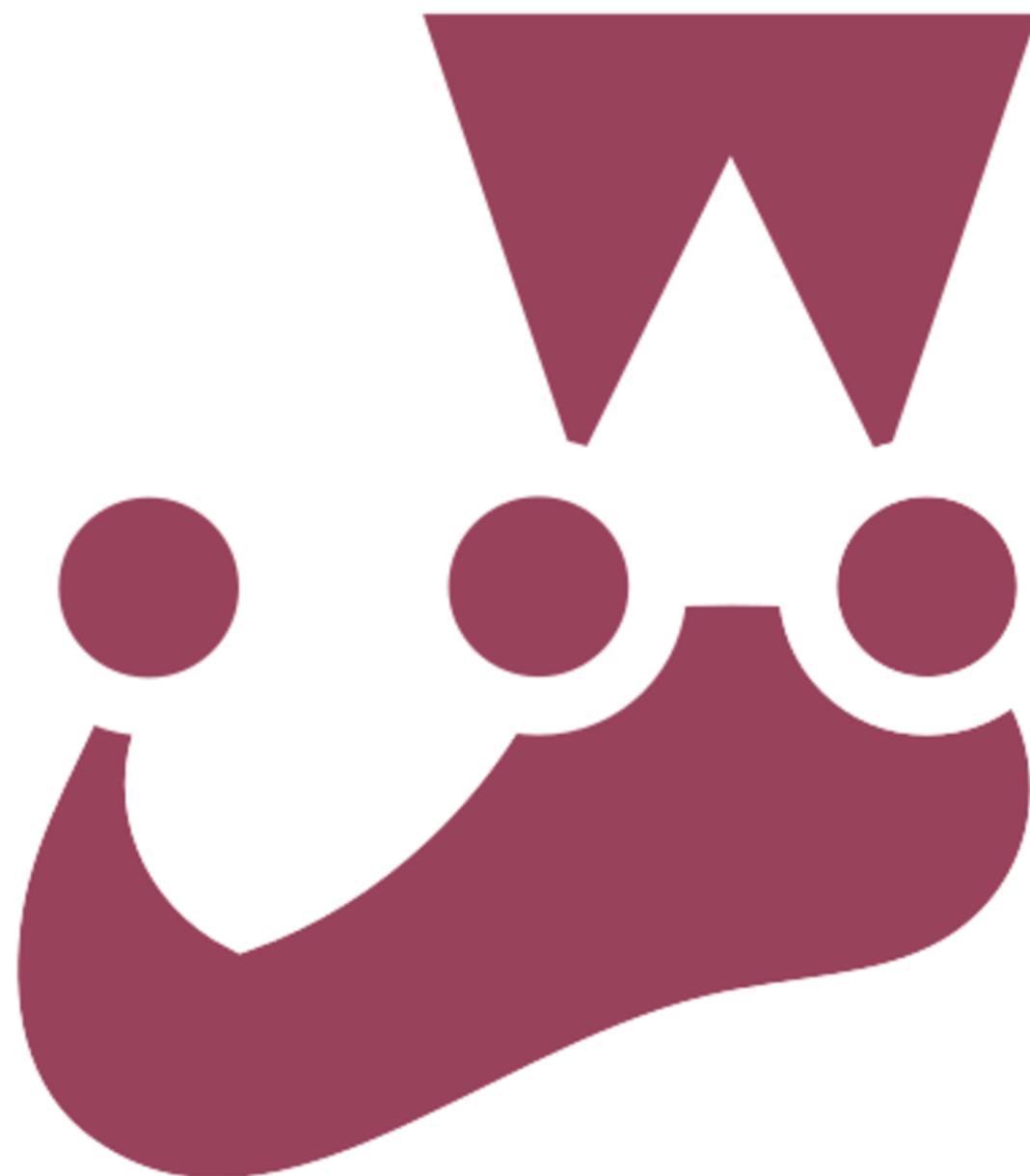
What is Jest?

- Testing Library/Framework
- Comparable to XUnit
 - Command Line Runner
 - More Features



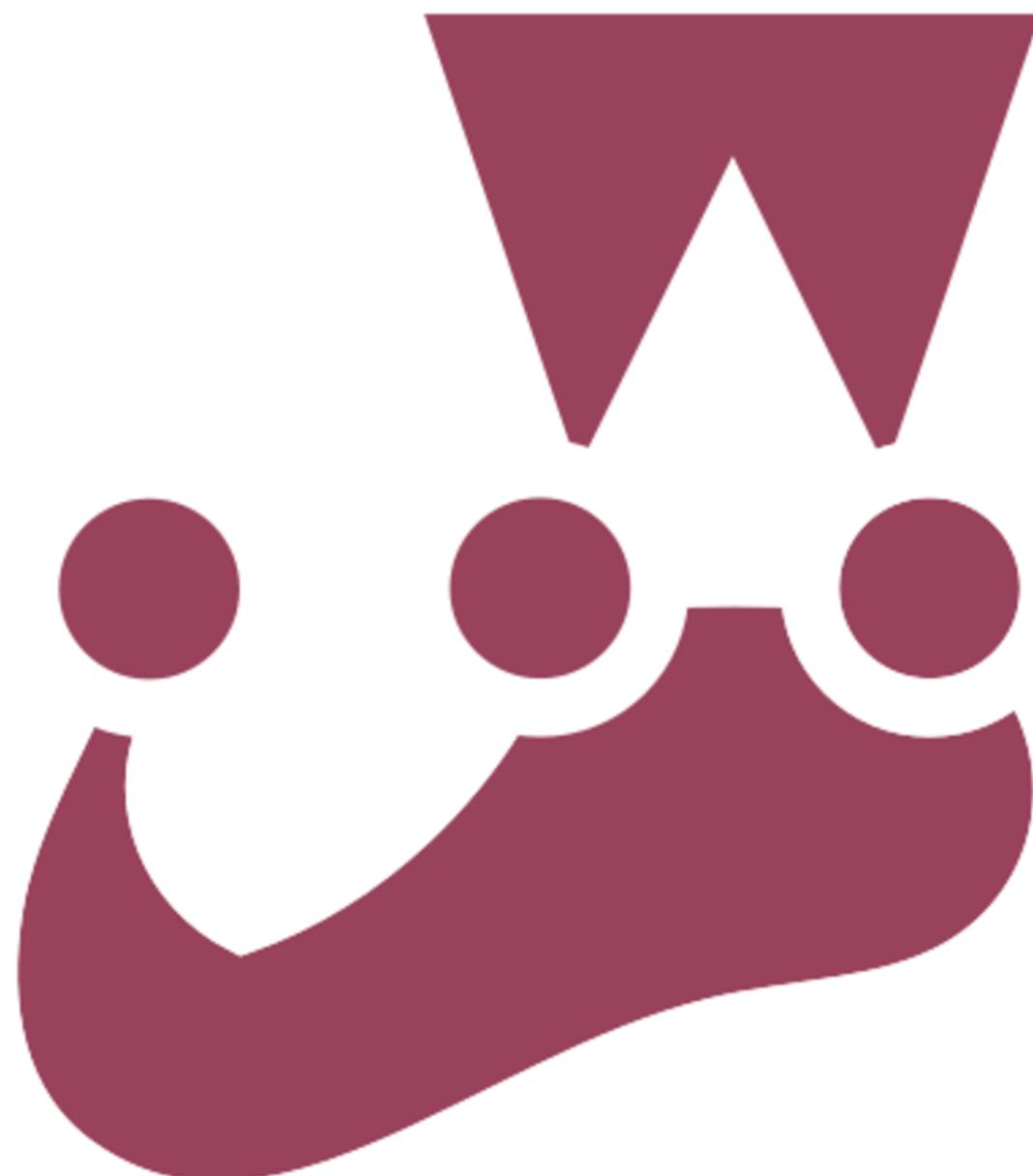
Jest: Facts and figures

- Developed by Facebook
- Current Version 22.4
- MIT Licensed
- > 16.000 Github Stars
- Has been around since 2014



Jest: Facts and figures

- Comes from Facebook but it's not limited to React
- Works like a charm in my projects
 - React based
 - Vue based
 - JQuery backed
 - Did not test it with Angular
 - But should work



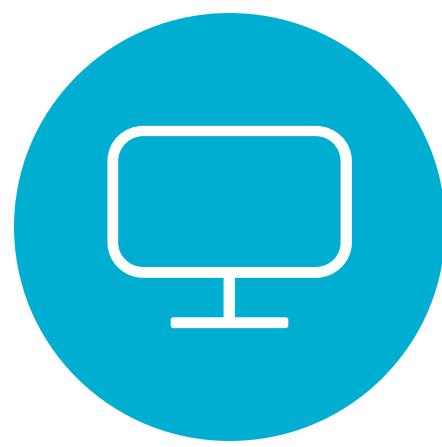
TOP 3 Reasons to use Jest now

You won't believe Number 2 :)



Easy Setup

You can start with very little config



Awesome CLI

A really polished product



All in the box

Little to no need for third party Libs

Easy setup

Simple to install and simple to use

BROOKS.

Let's get started

I want to write the first test!



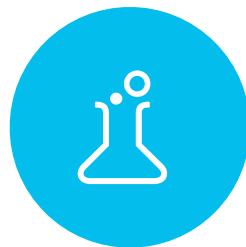
Create-React-App

Probably the easiest way to get going is to just use `create-react-app`



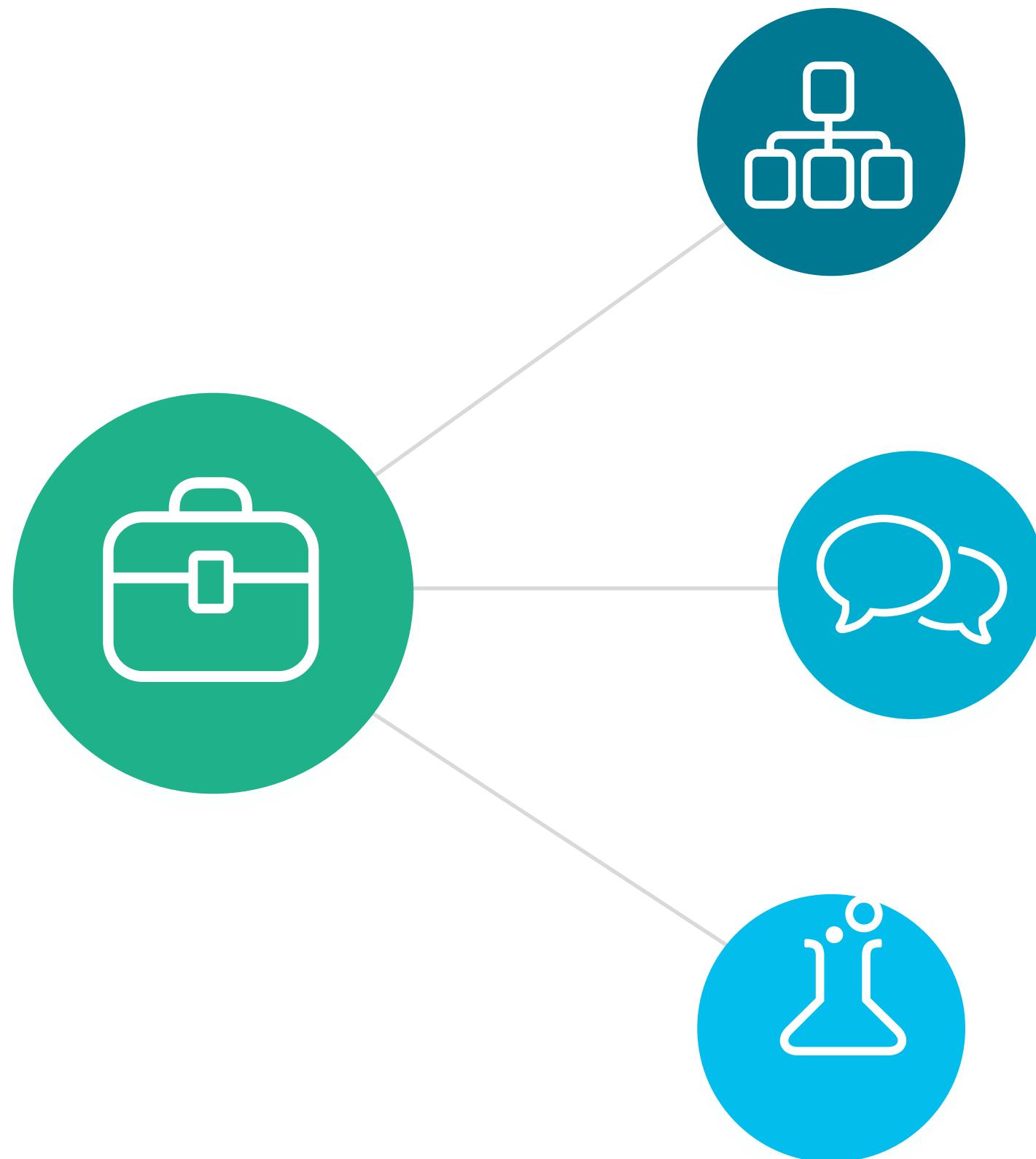
[repl.it](#)

Use an online repl might be even easier.



`npm install jest`

Come on let's get our hands dirty



Let's get started

🔊 Install dependencies

```
$ npm install -D jest
```

🔊 package.json

```
"scripts": {  
  "test": "jest",  
  "test:watch": "jest --watch"  
},
```

🚀 Write a simple test

```
describe('Demo should', () => {  
  it('be green', () => {  
    expect(true).toBe(true);  
  });  
});
```

⚙️ Run it

```
$> npm test  
  
PASS ./demo.spec.js  
Demo should  
  ✓ be green (2ms)  
  
Test Suites: 1 passed, 1 total  
Tests:       1 passed, 1 total  
Snapshots:   0 total  
Time:        1.606s  
Ran all test suites.
```

👍 Be amazed

Anatomy of a test

```
describe("add", () => {  
  beforeEach(() => {})  
  afterEach(() => {})  
  beforeAll(() => {})  
  afterAll(() => {})  
  
  it("should compute the sum", () => {  
    expect(add(1, 2)).toBe(3);  
  });  
});
```

This gives a nice descriptive scoped structure of your test.
As a bonus Jest can print the results nicely,
I can recommend that structure

```
test("should compute the sum", () => {  
  expect(add(1, 2)).toBe(3);  
})
```

A more traditional way of writing your tests
Possible, but less expressive than the describe style.

Where are my tests?

- Out of the box Jest looks for

```
_tests_/*  
foo.test.js  
bar.spec.js
```

- Configurable
- Suited my needs well thus far

Works with ES6

- Add babel

```
$ npm install -D babel-jest babel-core babel-preset-env
```

- Add a .babelrc

```
{  
  "presets": ["env"]  
}
```

- Jest picks it up automatically
- No need to configure
- Compile to JS friendly
 - e.g. works (now) well with TypeScript

Configurable

- in your package.json

```
"jest": {  
  "verbose": true  
}
```

- via JavaScript

```
// jest.config.js  
module.exports = {  
  verbose: true,  
};
```

- Or using CLI

- <https://facebook.github.io/jest/docs/en/configuration.html>

Sandboxed and parallel

- Runs errors tests first
- Runs tests in parallel
- Tests run in a sandbox
 - No conflicting

Optional: Install Jest globally

- Use Jest without configuring scripts
- I use it for convenience

```
$ npm install -g jest
```

Awesome CLI

Fun to work with

you
good
look.
d.

Test results

- Nice output out of the box

```
PASS  client/service/skippingAndForcing.spec.js
PASS  client/service/customMatchers.spec.js
PASS  client/service/asymmetricExpectations.spec.js
PASS  client/service/serviceUser.spec.js
PASS  client/service/newExpectations.spec.js
PASS  client/service/httpServiceWithPromises.spec.js
PASS  client/components/FirstTest.spec.js
PASS  client/components/TextComponent.spec.jsx
PASS  client/components/App.spec.jsx (5.539s)
PASS  client/componentsListComponent.spec.jsx (5.589s)
```

```
Test Suites: 10 passed, 10 total
Tests:      5 skipped, 23 passed, 28 total
Snapshots:   5 passed, 5 total
Time:        9.83s
Ran all test suites.
```

Meaningful error reports

```
$> jest
FAIL ./add.spec.js
• add > should compute the sum

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

Expected value to be:
  4
Received:
  3

  8 |
  9 |     it('should compute the sum', () => {
> 10 |       expect(add(1, 2)).toBe(4);
 11 |     });
 12 |   );
 13 | }

at Object.<anonymous> (add.spec.js:10:27)
```

```
PASS ./stringProducer.spec.js
PASS ./asymmetricExpectations.spec.js
...
PASS ./skippingAndForcing.spec.js
```

```
Test Suites: 1 failed, 8 passed, 9 total
Tests:       1 failed, 5 skipped, 15 passed, 21 total
Snapshots:   1 passed, 1 total
Time:        3.53s
Ran all test suites.
```

Watch mode

- Nice for TDD



1. fish /Users/hgp/development/sideprojects/talks/jestStarter (fish)

Code coverage

- Computes coverage out of the box

```
$ jest --coverage
PASS  client/service/newExpectations.spec.js
```

```
Test Suites: 10 passed, 10 total
Tests:      5 skipped, 23 passed, 28 total
Snapshots:   5 passed, 5 total
Time:        4.099s
Ran all test suites.
```

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Lines
All files	90.91	100	86.67	93.33	
components	100	100	100	100	
App.jsx	100	100	100	100	
ListComponent.jsx	100	100	100	100	
TextComponent.jsx	100	100	100	100	
service	81.25	100	77.78	84.62	
httpService.js	100	100	100	100	
httpServiceWithPromises.js	100	100	100	100	
serviceUser.js	62.5	100	50	66.67	9,10

Even nicer output

- verbose Option

```
$ jest --verbose
PASS  client/components/App.spec.jsx
  App
    ✓ renders (14ms)
    ✓ calling Service correctly (3ms)
    ✓ renders with defined data (1ms)
    ✓ renders with defined return function (1ms)

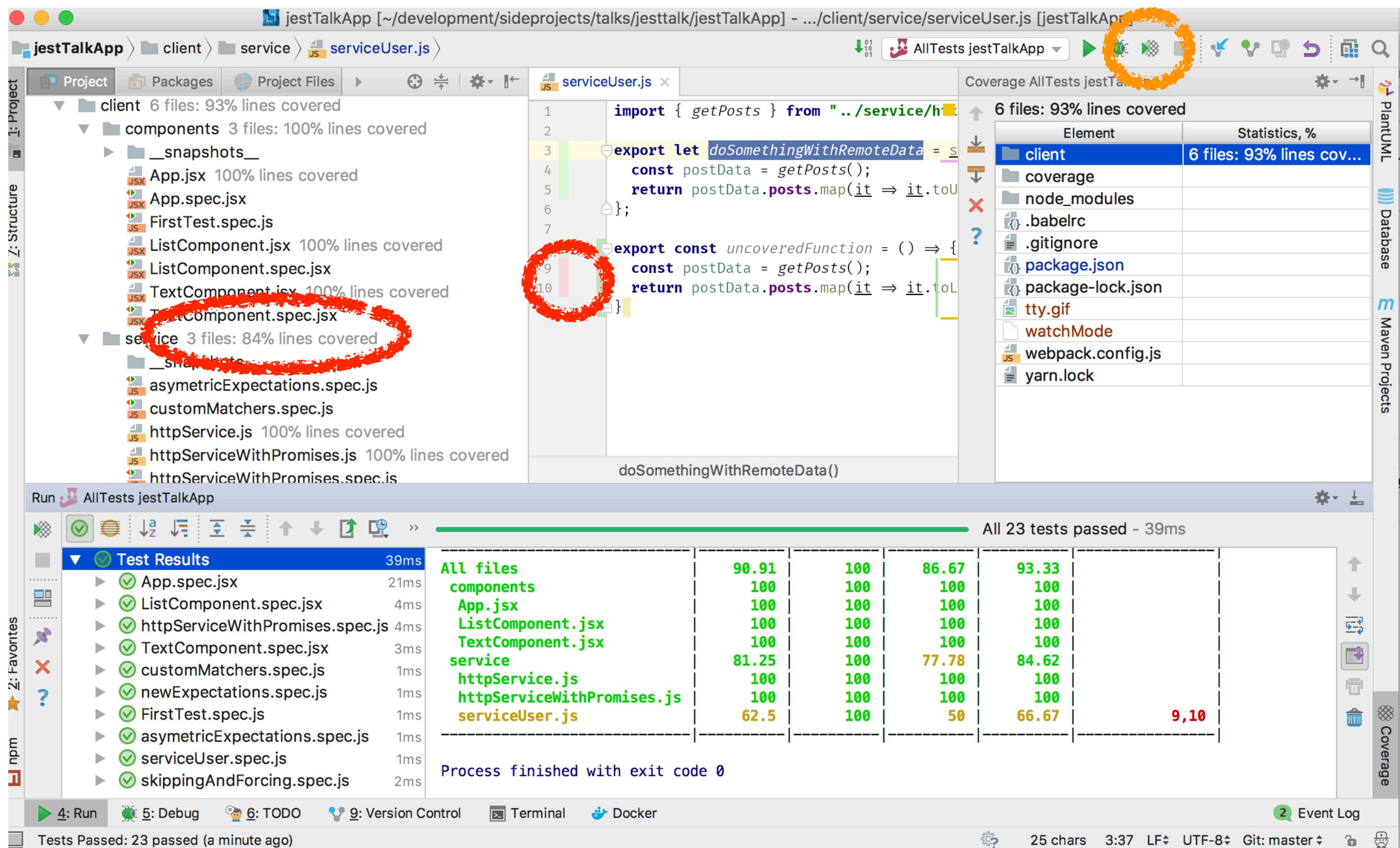
PASS  client/service/httpServiceWithPromises.spec.js
  httpService getPosts should
    ✓ talk to the backend using done (1ms)
    ✓ talk to the backend using promise returns
    ✓ talk to the backend using expectations (1ms)
    ✓ talk to the backend using async await (1ms)
  httpService getPostsRejecting should
    ✓ reject using done
    ✓ reject using promise returns (1ms)
    ✓ reject using expectations
    ✓ reject using async await (1ms)
```

Test Suites: 10 passed, 10 total
Tests: 5 skipped, 23 passed, 28 total
Snapshots: 5 passed, 5 total
Time: 2.187s
Ran all test suites.

There's more

- `--onlyChanged / -o`
 - runs test affected by files changed
- `--lastCommit`
 - runs test affected by files changed in the last commit

Editor integration IntelliJ



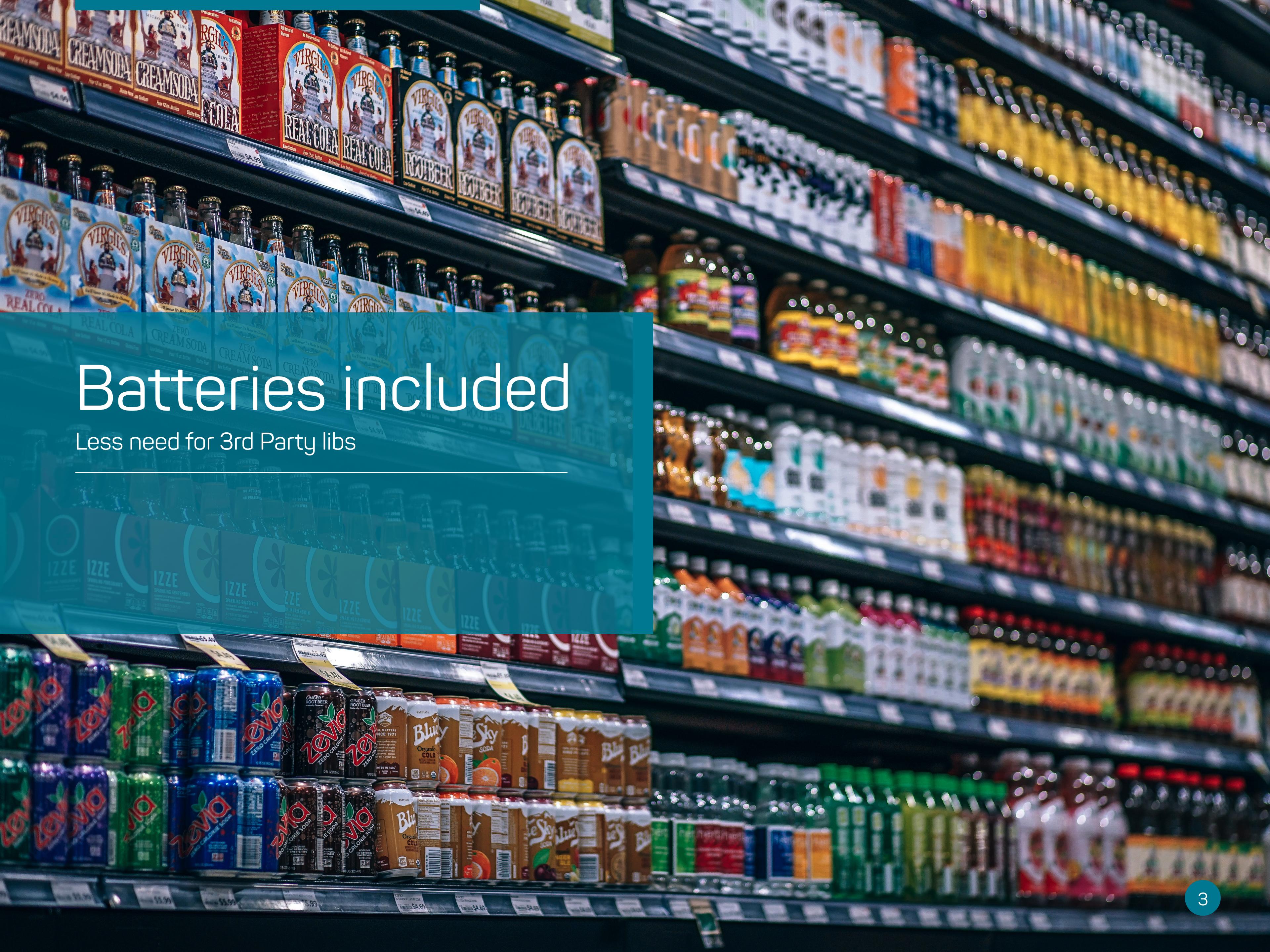
IDE Support

It just works

- I use it mainly in IntelliJ/Webstorm and it just works
 - Use a Run Config like any other test
 - In EAP even coverage is shown
- Works fine in VSCode
 - Some more config
 - Not that polished than IntelliJ
 - But fine for me
 - I use the Jest extension
- Did not test it with Eclipse
- I often use it on the command line
 - Alongside the IDE
 - The watch feature shines there

Batteries included

Less need for 3rd Party libs



What's in the box



Expectations

- No need for separate Mocha/Jasmine/expect
- All the expectations you need
- <https://facebook.github.io/jest/docs/en/expect.html>

.[toBe\(value\)](#)
.a
.[toHaveBeenCalled\(\)](#)
.a
.[toBeCloseTo\(number, numDigits\)](#)
.a
.[toBeDefined\(\)](#)
.a
.[toBeFalsy\(\)](#)
.a
.[toBeGreaterThan\(number\)](#)
.a
.[toBeGreaterThanOrEqual\(number\)](#)
.a
.[toBeLessThan\(number\)](#)
.a
.[toBeLessThanOrEqual\(number\)](#)
.a
.[toBeInstanceOf\(Class\)](#)
.a
.[toBeNull\(\)](#)
.a
.[toBeTruthy\(\)](#)
.a
.[beUndefined\(\)](#)
.a
.[toContain\(item\)](#)
.a
.[toContainEqual\(item\)](#)
.a
.[toEqual\(value\)](#)
.a
.[toHaveLength\(number\)](#)
.a
.[toMatch\(regexpOrString\)](#)
.a
.[toMatchObject\(object\)](#)
...

Expectations: Examples

Expecting Objects

```
export const objectProducer = (index) => ({
  booleanProperty: (index % 2) === 0,
  stringProperty: 'test' + index
});
```

```
describe('object producer should', () => {
  it('produce an object', () => {
    const object = objectProducer(1);
    expect(object).toBeDefined();
    expect(object.booleanProperty).toBe(false);
    expect(object.stringProperty).toEqual("test1");
    expect(object.stringProperty).toEqual(expect.stringContaining('est'));
    expect(object).toEqual(expect.objectContaining({booleanProperty: false}));
  });
});
```

Expecting Arrays

```
export function arrayProducer() {
  return [objectProducer(1), objectProducer(3), objectProducer(3), objectProducer(4)];
}
```

```
describe('array producer should', () => {
  it('produce an array', () => {
    const array = arrayProducer();
    expect(array).toContainEqual({stringProperty: "test2", booleanProperty: true});
  });
});
```

Useful shortcuts

xdescribe()/describe.skip()

xit()/it.skip()

xtest()/test.skip()

Skips the block

fdescribe()/describe.only()

fit()/it.only()

ftest()/test.only()

Executes only this block (in the block)

Matchers are extendable

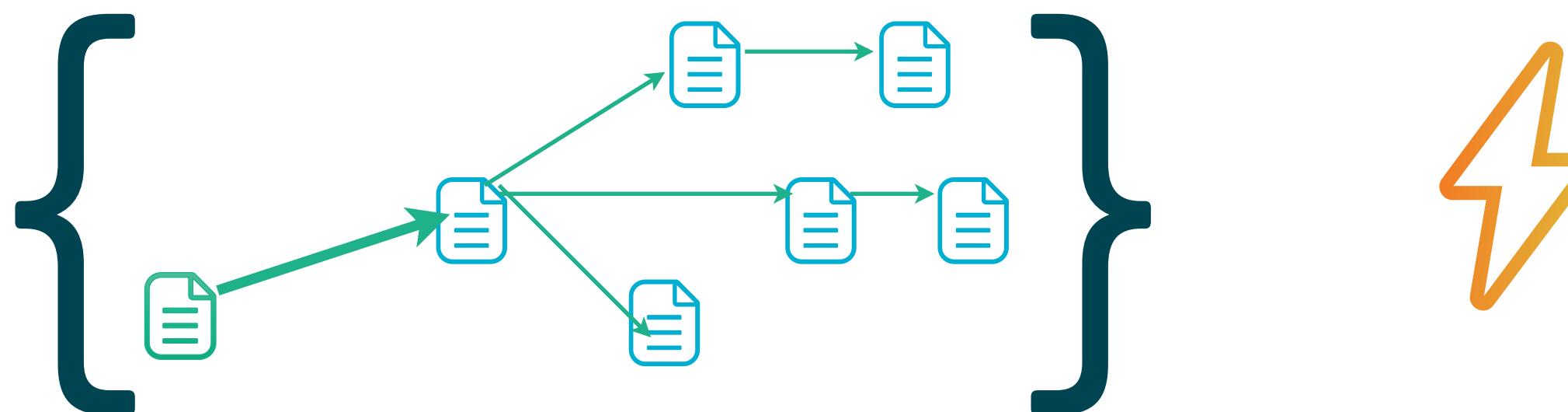
- To make your tests better readable

```
expect.extend({
  arrayLengthDivisibleBy(received, argument) {
    const pass = received.length % argument == 0;
    if (pass) {
      return {
        message: () =>
          `expected ${received} not to be divisible by ${argument}`,
        pass: true,
      };
    } else {
      return {
        message: () => `expected ${received} to be divisible by ${argument}`,
        pass: false,
      };
    }
  },
});
```

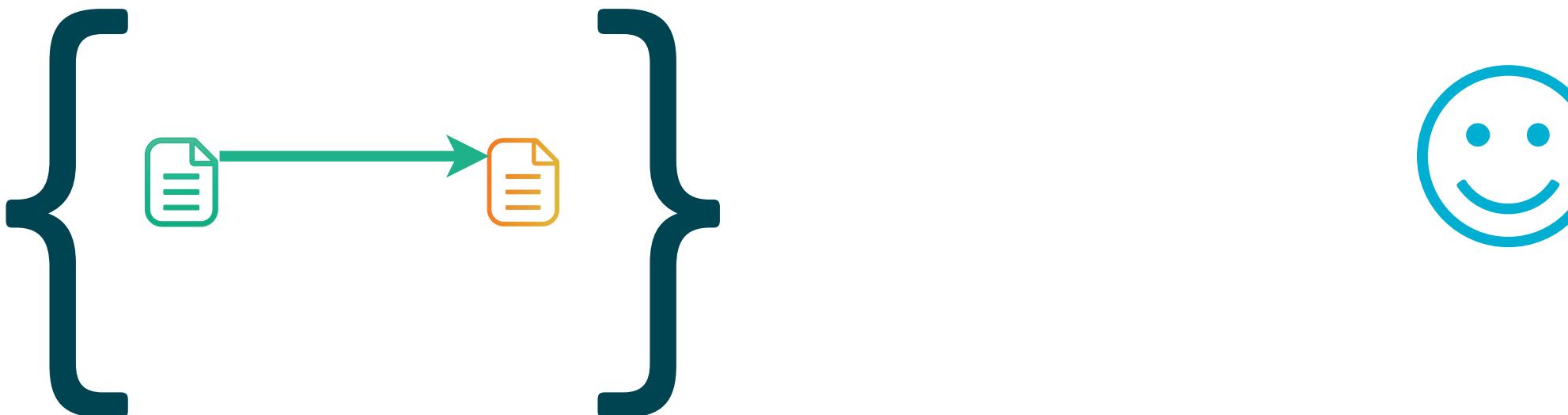
```
describe('array producer should', () => {
  it('produce an array with even number', () => {
    const array = arrayProducer();
    expect(array).arrayLengthDivisibleBy(2);
  });
});
```

Mocking

In a unit test it is often unfeasible to kickstart your whole dependency graph



Mocking lets you narrow down the scope of your test



Mocking

- No need for separate Sinon.js
- Works similar to Mockito in Java



Mock Functions

Lets you replace more complex logic with logic right for your test



Manual Mocks

Lets you replace whole modules for your test

Mock functions

- Mock calls

```
export function objectCaller(toBeCalled) {  
    toBeCalled.fire('1');  
    toBeCalled.fire('2');  
    toBeCalled.fire('3');  
}
```

```
it('call collaborator', () => {  
    const collaborator = {  
        fire: jest.fn()  
    };  
    objectCaller(collaborator);  
    expect(collaborator.fire).toHaveBeenCalled();  
    expect(collaborator.fire).toHaveBeenCalledTimes(3);  
    expect(collaborator.fire).lastCalledWith('3');  
    expect(collaborator.fire).toHaveBeenCalledWith('2');  
    expect(collaborator.fire.mock.calls[1][0]).toBe('2');  
});
```

- Mock returns

```
export function objectCallerAndReturner(toBeCalled) {  
    return toBeCalled.trigger('1');  
}
```

```
it('return data from collaborator', () => {  
    const collaborator = {  
        trigger: jest.fn().mockReturnValue('22')  
    };  
    objectCallerAndReturner(collaborator);  
    expect(collaborator.trigger()).toBe('22');  
});
```

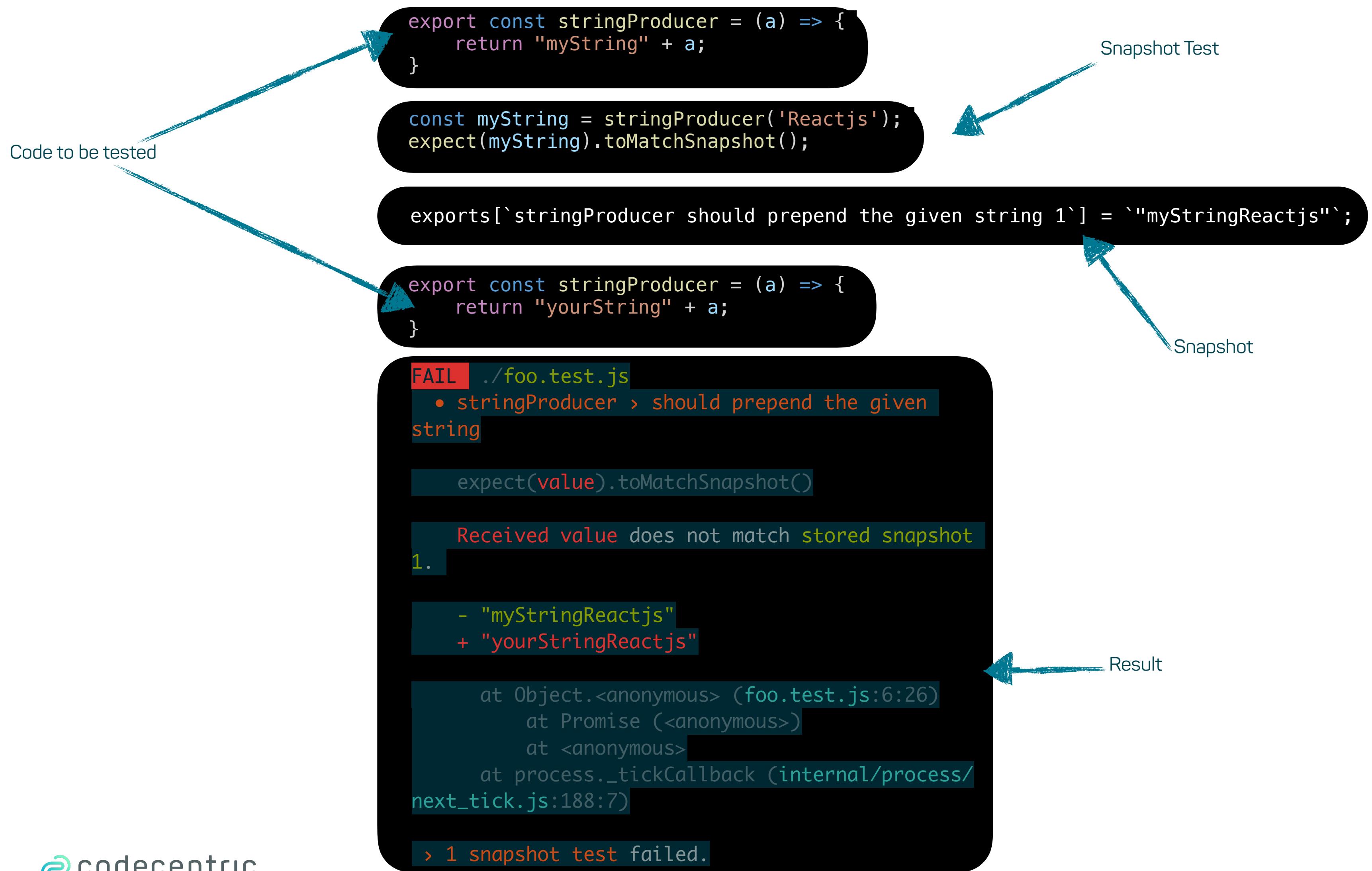
Mock a module

```
import { doSomethingWithRemoteData } from "./serviceUser.js";

jest.mock("./httpService", () => {
  const getPostMock = () => {
    return {
      posts: ["fake", "data"]
    };
  };
  return {
    getPosts: getPostMock
  };
});

describe("serviceUser should", () => {
  it("talk to backendService", () => {
    const data = doSomethingWithRemoteData();
    expect(data).toEqual(["FAKE", "DATA"]);
  });
});
```

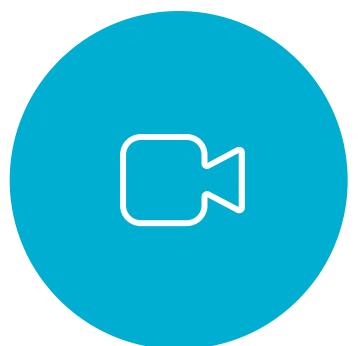
Snapshot tests by example



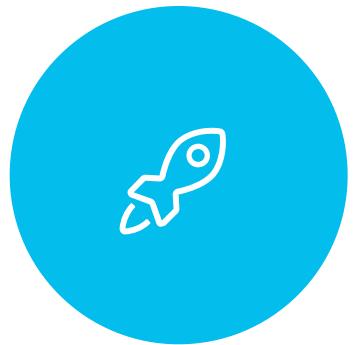
So what is this useful for?



Test that would require some effort to expect



React components (that should not change)



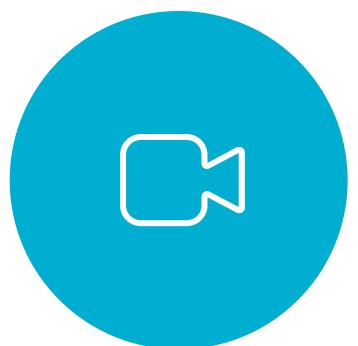
Things you would not test otherwise

That being said

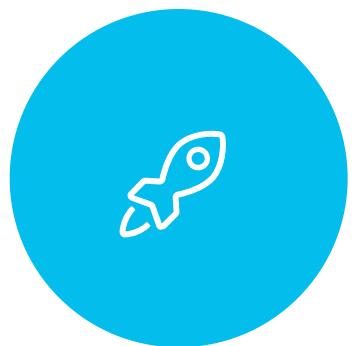
Besides snapshot tests being awesome



Snapshot test may be brittle



You NEED to review your snapshot changes
No jest -u fire and forget



Prefer unit tests

DOM Testing

- Jest comes with JSDOM to support testing DOM.
- JSDOM simulates a DOM-environment.

```
const setupDOM = () => {
  document.body.innerHTML =
    '<div>' +
    '  <div id="openDialog" >Click</div>' +
    '  <div id="dialog" />' +
    '' +
    '</div>';
  initPage();
};
```

```
it('open on a click at the clickable element', () => {
  document.querySelector('#openDialog').click();
  expect(isDialogOpen()).toBe(true);
});
```

Async Tests

- Are supported out-of-the box

```
describe("httpService getPosts should", () => {
  const response = {
    posts: ["complicated", "JSON", "Response"]
  };

  it("talk to the backend using done", done => {
    getPosts().then(data => {
      expect(data).toEqual(response);
      done();
    });
  });

  it("talk to the backend using promise returns", () => {
    return getPosts().then(data => {
      expect(data).toEqual(response);
    });
  });

  it("talk to the backend using expectations", () => {
    expect(getPosts()).resolves.toEqual(response);
  });

  it("talk to the backend using async await", async () => {
    const posts = await getPosts();
    expect(posts).toEqual(response);
  });
});
```



- I like the `async/await` Syntax

What's in the box



Coming to an end

Some closing words



What we had to do

🔊 Install dependencies

```
$ npm install -D jest
```

🔊 package.json

```
"scripts": {  
  "test": "jest",  
  "test:watch": "jest --watch"  
}
```

🔊 babel

```
$ npm install -D babel-jest babel-core babel-preset-env
```

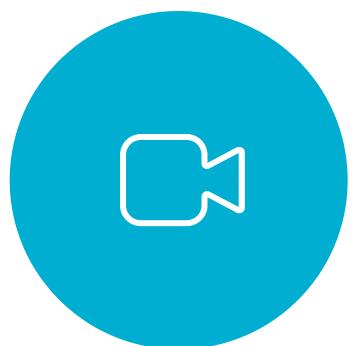
👓 <https://github.com/holgergp/jestStarter>

```
/.babelrc  
{  
  "presets": ["env"]  
}
```

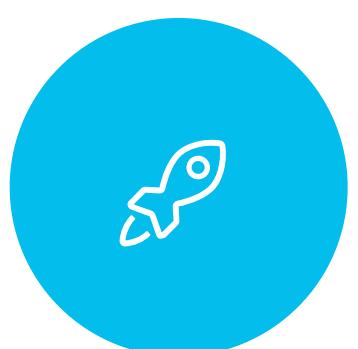
Try Jest yourself



Easy to use and setup



Powerful features set yet familiar



Polished product (even the CLI)

**START
TESTING
YOUR
JS
TODAY**



Tryout Jest

<https://facebook.github.io/jest/>

Thanks!





A photograph showing a person's hands raised in a dark room. The hands are silhouetted against a bright background of blurred lights. One hand wears a large, ornate watch on the wrist. The other hand has a ring on the middle finger. A teal-colored rectangular overlay covers the top left portion of the image, containing the text "Questions?"

Questions?

Picture Links

- <https://unsplash.com/photos/FQjmQgSoRyQ>
- <https://unsplash.com/photos/gYxVSeZazXU>
- <https://unsplash.com/photos/h1v8IkfDUu4>
- <https://unsplash.com/photos/58tOB36ZTnw>
- <https://unsplash.com/photos/cEUI-ODSM9s>
- <https://unsplash.com/photos/9HI8UJMSdZA>
- <https://unsplash.com/photos/xhWMi9wdQaE>
- <https://unsplash.com/photos/-3wygakaeQc>
- <https://unsplash.com/photos/R6xx6fnvPT8>
- <https://unsplash.com/photos/Q6jX7BbPE38>
- <https://unsplash.com/photos/8xAAOf9yQnE>
- <https://unsplash.com/photos/OYbeoQOX89k>
- <https://unsplash.com/photos/RpDA3uYkJWM>
- <https://unsplash.com/photos/6dvxVmG5nUU>
- <https://unsplash.com/photos/D56TpVU8zng>
- <https://unsplash.com/photos/TyQ-OlPp6e4>
- <https://unsplash.com/photos/gdBXILO53N4>

Stay connected

Thanks for listening!

Our mission – to promote agile development, innovation and technology – extends through everything we do.



Address

codecentric AG
Hochstraße 11
42697 Solingen



Contact Info

E-Mail: info@codecentric.de
www.codecentric.de



Telephone

Telefon: +49 (0) 212. 23 36 28 0
Telefax: +49 (0) 212.23 36 28 79

