Jest: Frontend Testing done right

Holger Grosse-Plankermann

codecentric

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

codecentric

Show of hands

Do you test your frontend code?

- Unit Testing
- Selenium/Test Cafe etc.
- HP Quality Center etc
- Macros

Show of hands

What test framework are you using?

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

AGENDA

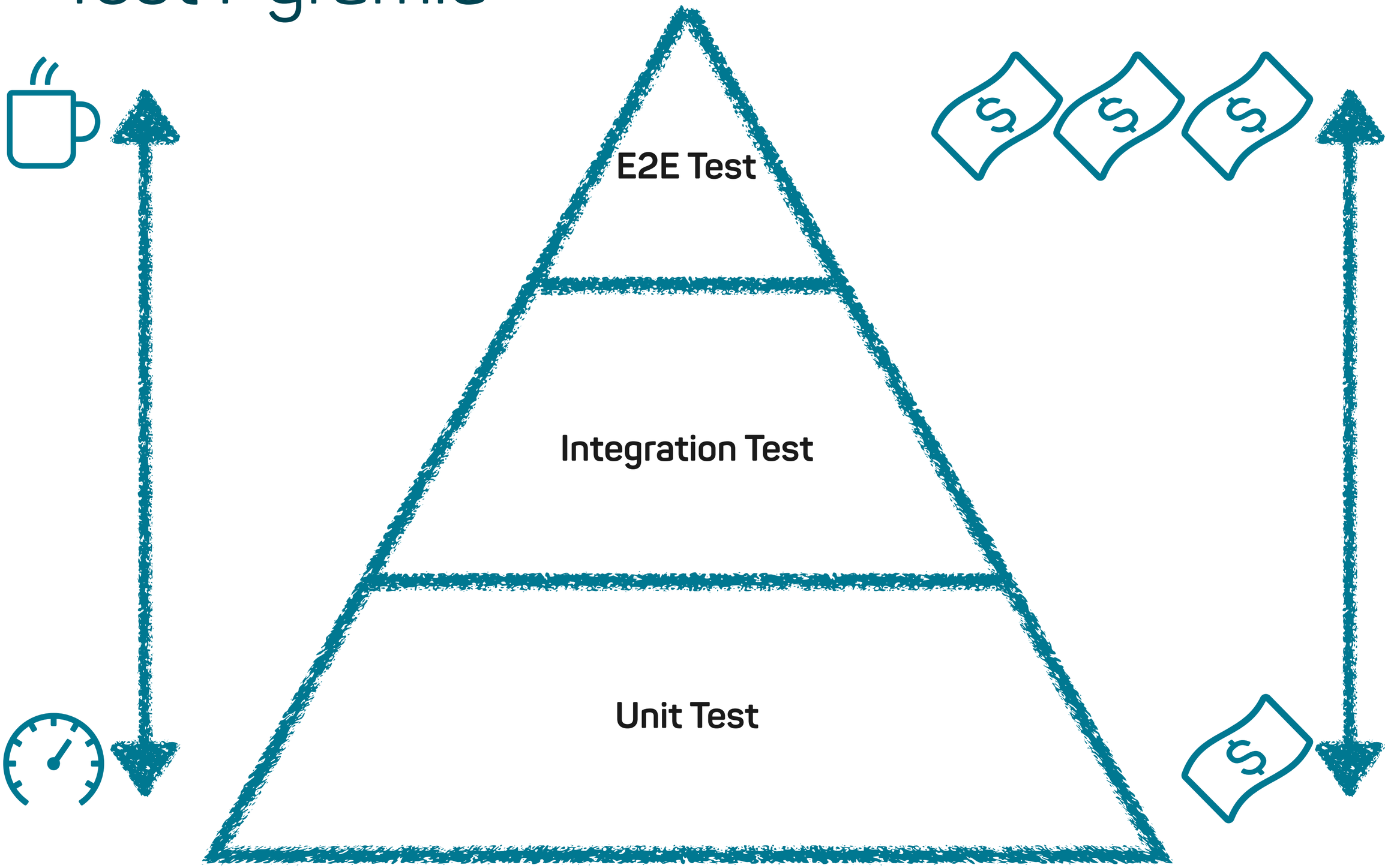
Where are we heading?

- (Unit-) Tests in JavaScript
  Where do I come from?
- What is Jest
  Why another library?
- Top 3 reasons for Jest
  Why I think Jest is awesome!

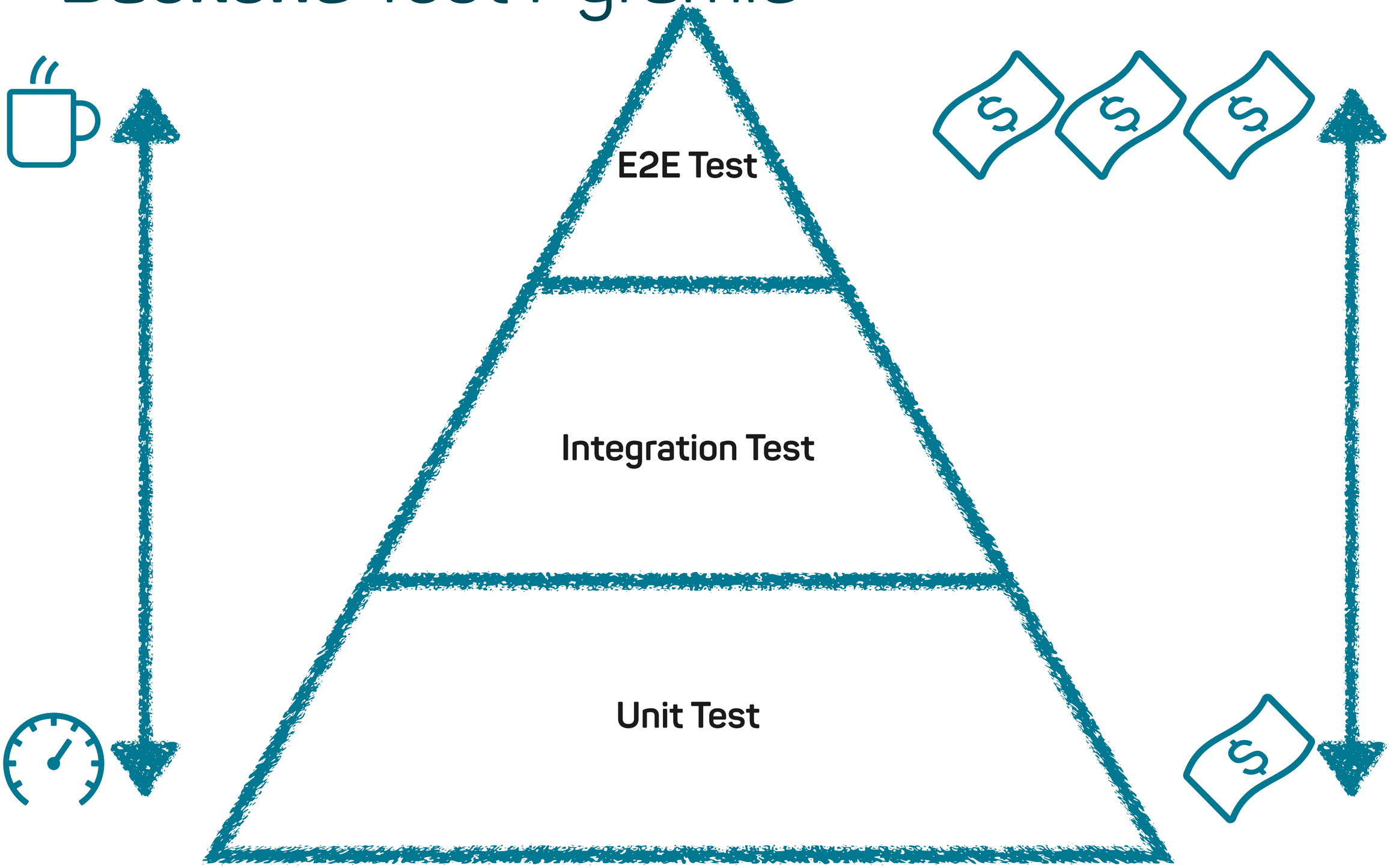
(Unit-) Testing JS

Where do I come from

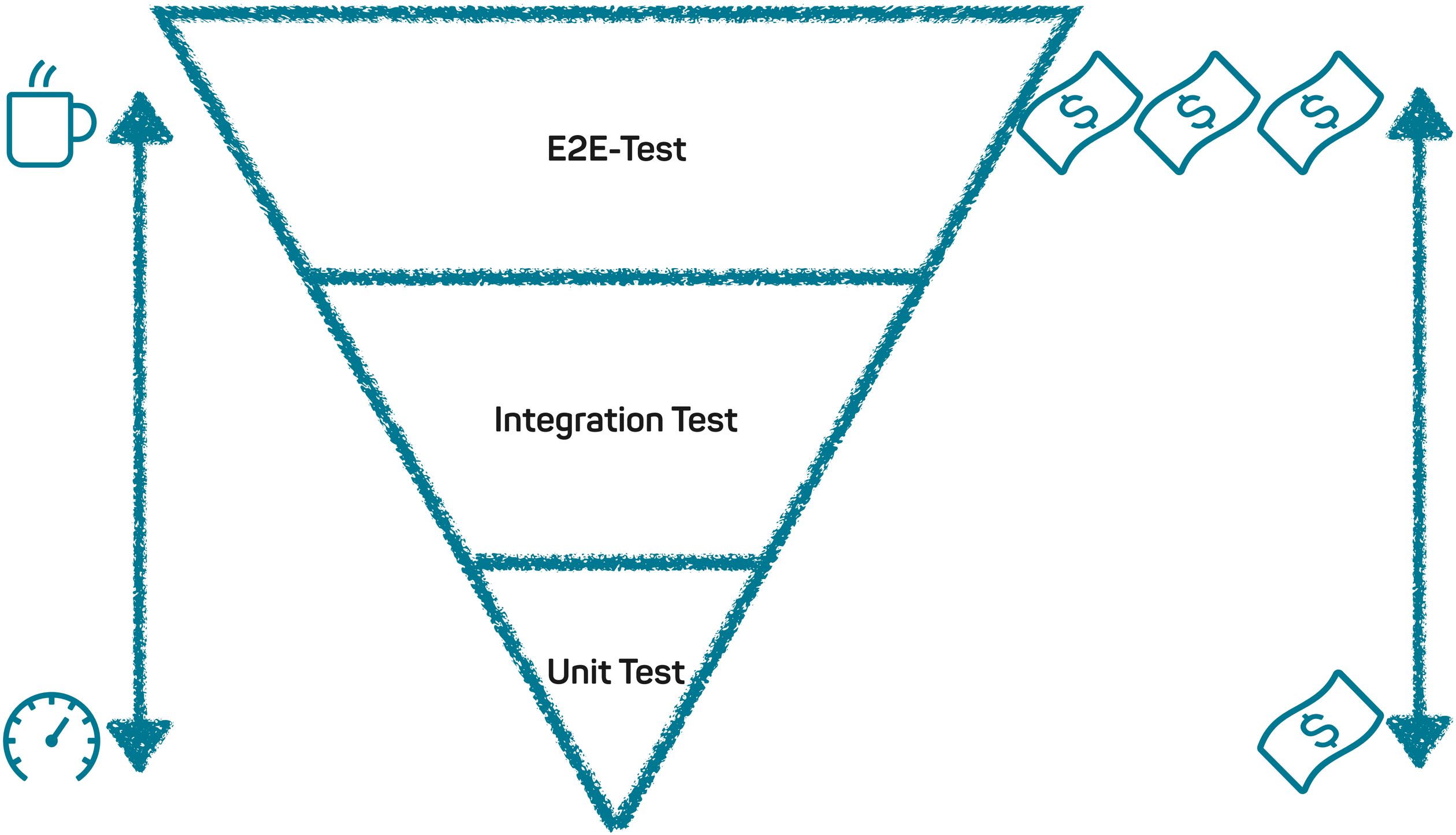
Test Pyramid



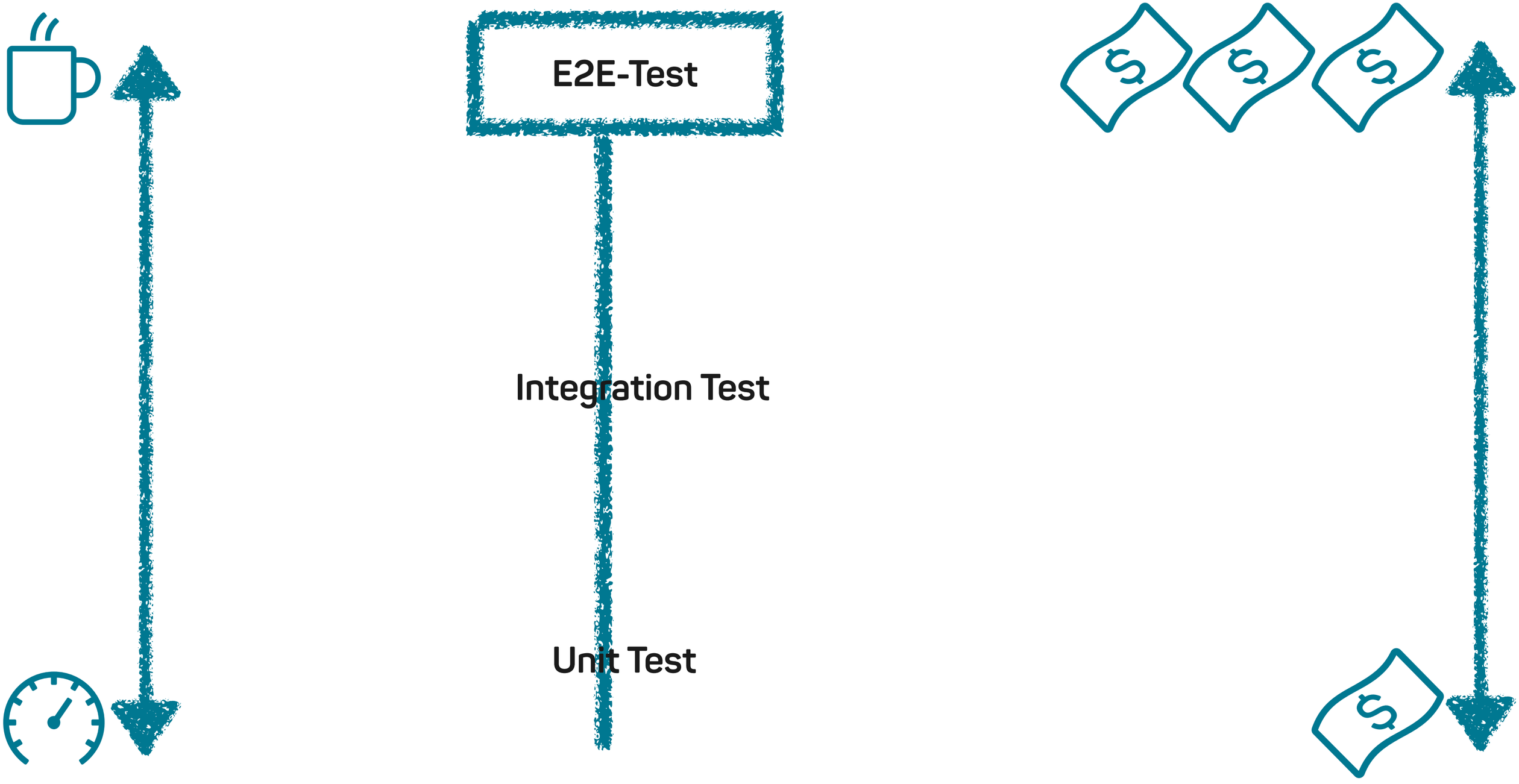
Backend Test Pyramid



Frontend Test Pyramid



Frontend Test Pyramid



In the backend

- Backend (Java) testing is well established
  - The important stuff is in the backend!
    Some enterprise architect
  - Need for unit testing
  - And tooling is quite nice tbh
    - mvn clean install ftw

In the frontend

- Frontend (JS) testing used to be an afterthought
  - Less (underrated!) complexity
    - Nah! We don't need to test that

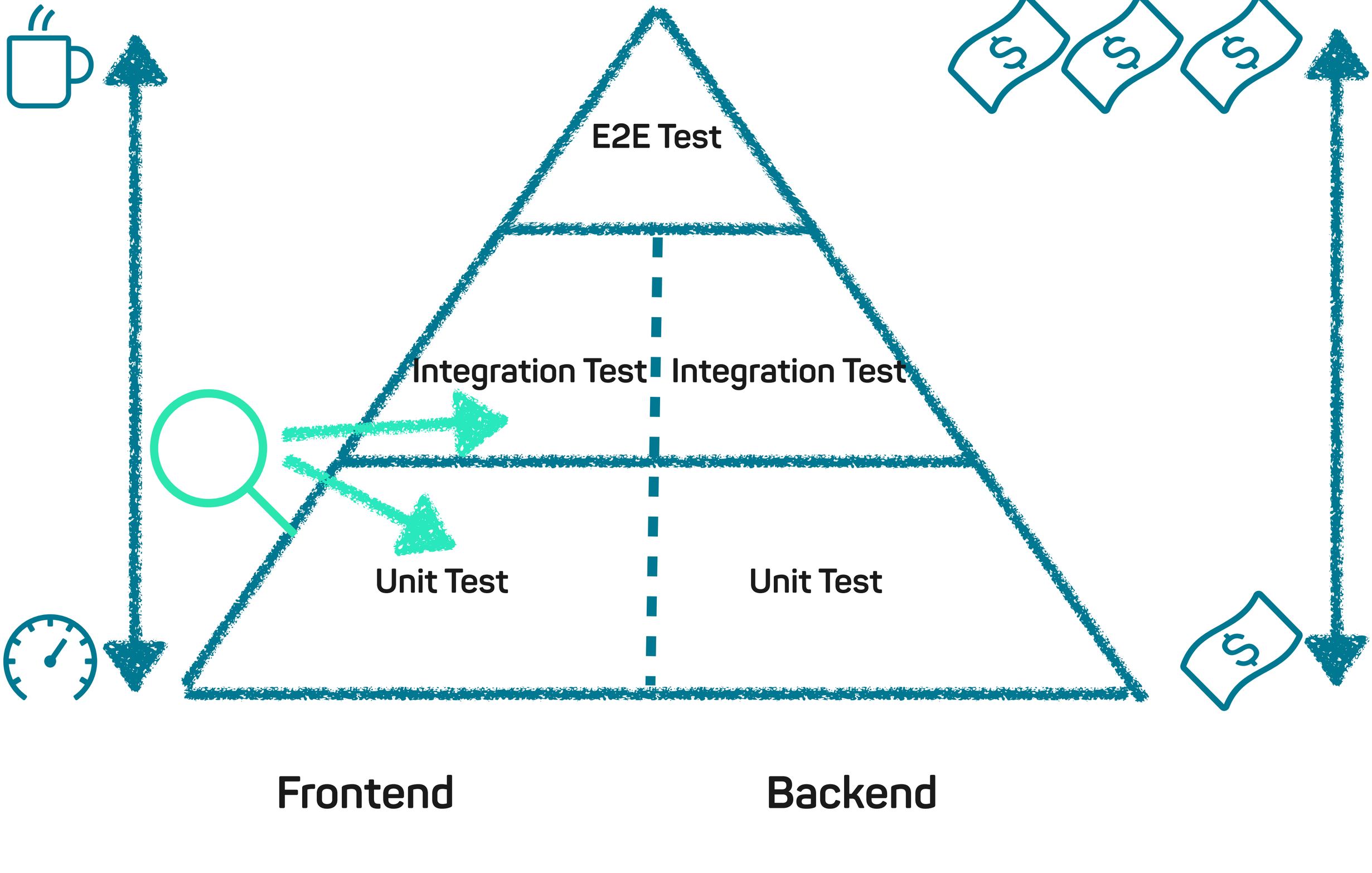
      The same enterprise architect
  - Messy tooling
    - Tons of config and dependencies needed to be orchestrated
- And that rumor is still in our heads

Proper Testing in the frontend is important

- In the age of SPAs
- More code than in the past
- Complex code
    - Frontend code in itself is complex
- Also: Compared to backends

- Need for testing is already there
    - Selenium is not enough!

Two sided test pyramid



Enter Jest

A library that puts the fun in testing

What is Jest?

Delightful JavaScript Testing

http://jestjs.io/

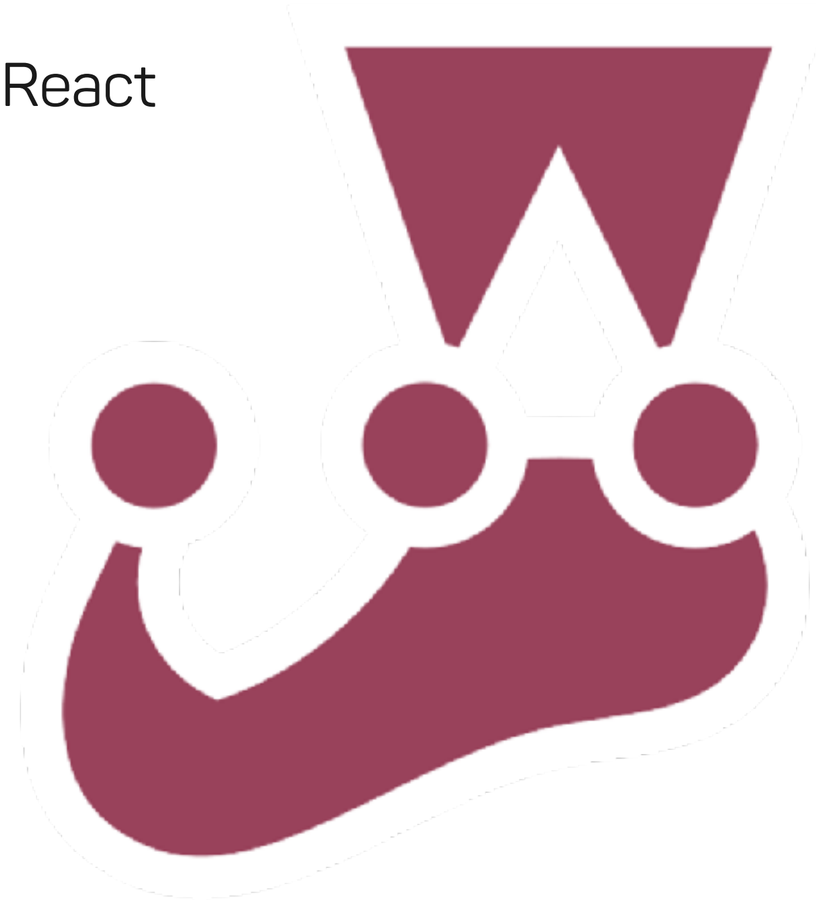
- Testing library/framework
    - node based
- Comparable to
    - JUnit (Java)
    - NUnit (.NET)
    - Test::Unit (Ruby)
    - PHPUnit (PHP)

Jest: Facts and figures

- Developed by Facebook
- Current version 23
- MIT licensed
- > 18.000 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
    - https://github.com/vuejs/vue-jest
  - JQuery backed
  - Angular
    - https://github.com/thymikee/jest-preset-angular



Why I like Jest

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

Let's get started

I want to write the first test!

Create-React-App

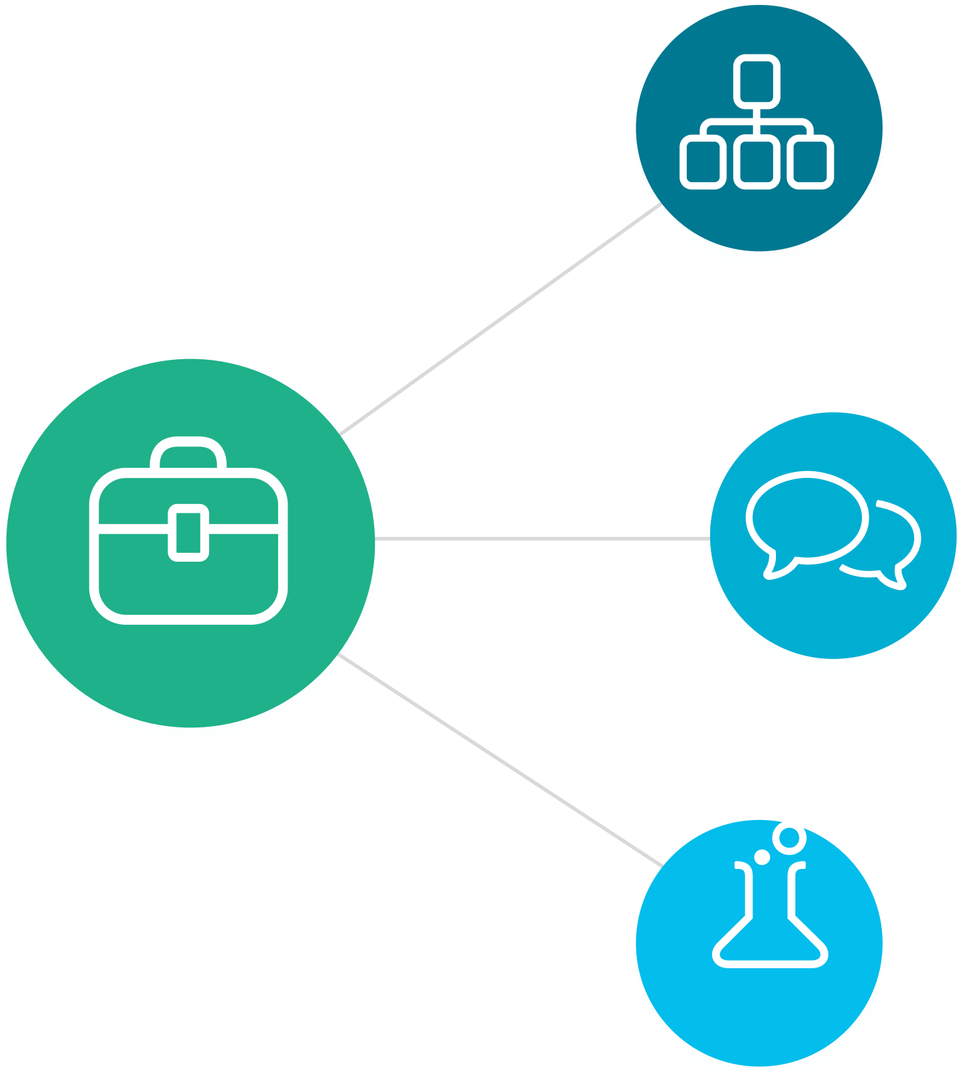
Probably the easiest way to get going is to just use create-react-app.

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
- package.json
- Write a simple test
- Run it
- Be amazed

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

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

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

```
$> 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.
```

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.

Best practice: Nested blocks

- Use nesting of describe blocks to group your tests logically

- Setup code can be fine tuned for specific case
  - Setup in nested describe

```
describe('Customer', () => {
    beforeEach(() => {});

    describe('placeOrder should', () => {
        beforeEach(() => {});
        it('be valid', () => {
            expect(customer.placeOrder()).toBe(true);
        });
    });

    describe('checkQty should', () => {
        beforeEach(() => {});
        it('be valid', () => {
            expect(customer.checkQty()).toBe(1);
        });
    });
});
```

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)

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

Best practice: Code and Test nearby

- Makes it way easier to navigate between code and test
- Unusual at first, but grow fond of it

```
src/myApp
    add.js
    add.spec.js
    mockable.js
    mockable.spec.js
    objectProducer.js
    objectProducer.spec.js
    stringProducer.js
    stringProducer.spec.js
```

There is more

- Works with ES6
    - With Babel
- Compile to JS friendly
    - e.g. works (now) well with TypeScript
- All presets are configurable
    - package.json
    - CLI
    - JS

Test runner trivia

- Jest runs
  - tests in parallel
  - tests in a sandbox
    - No conflicting of tests
  - failed tests first

Runs in your CI Server

I work with it in:

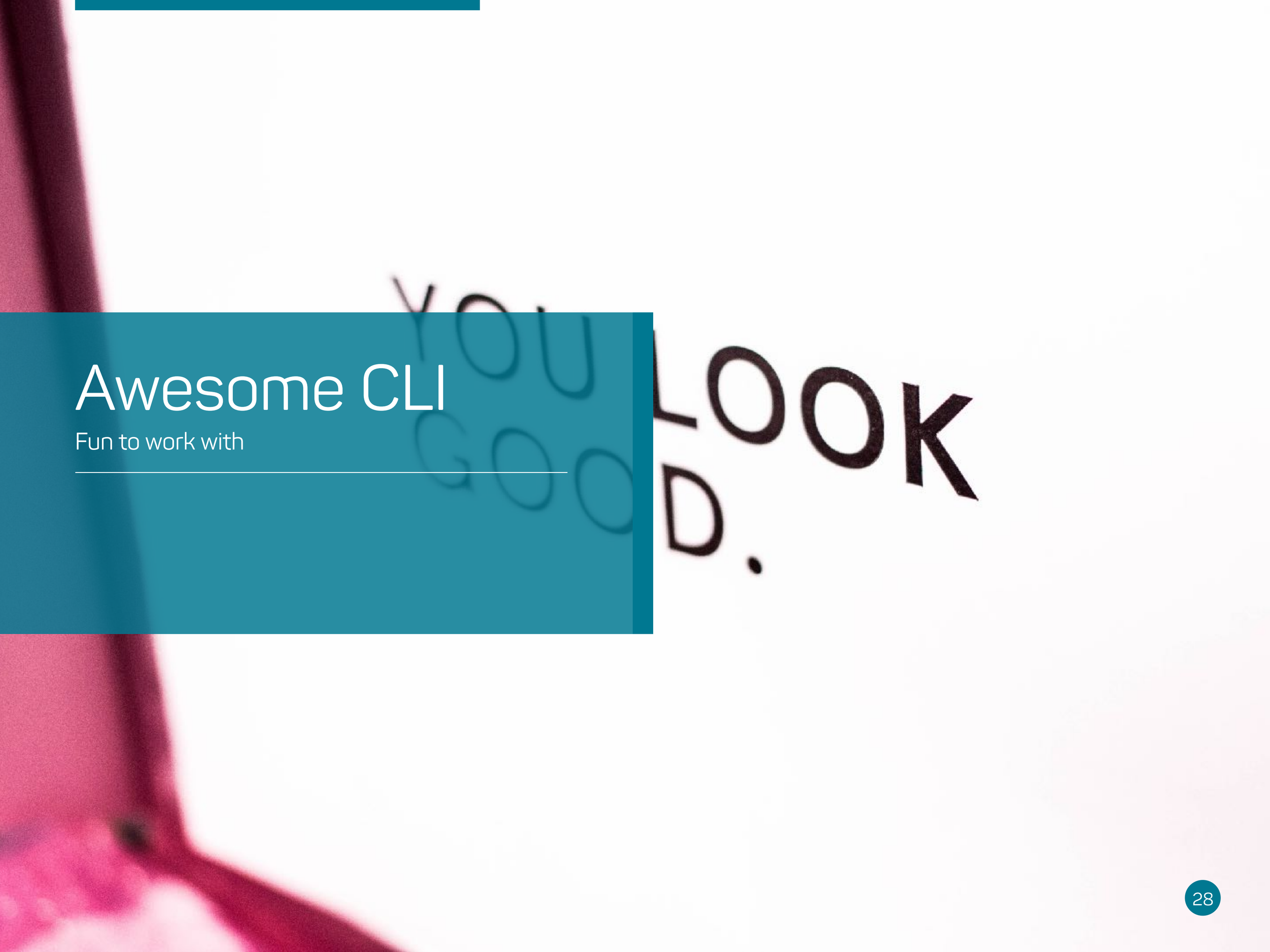
- Travis

- Gitlab CI

- Jenkins

Awesome CLI

Fun to work with



Test results

- Nice output out of the box

```
 PASS  client/service/skippingAndForcing.spec.js
 PASS  client/service/customMatchers.spec.js
 PASS  client/service/asymetricExpectations.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/components/ListComponent.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

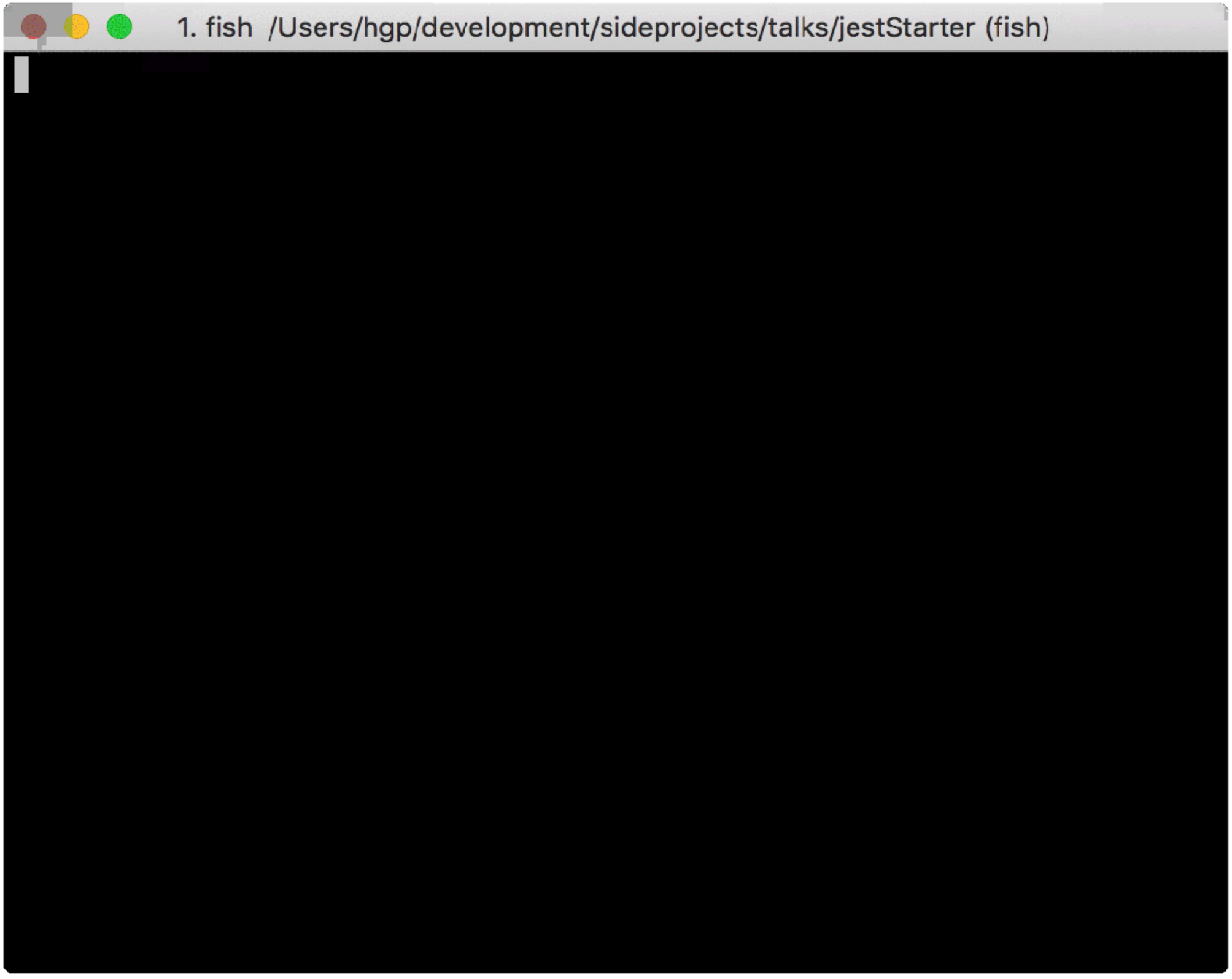
```
FAIL  ./add.spec.js
  • add › should compute the sum

    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  ./asymetricExpectations.spec.js
..

Test Suites: 1 failed, 8 passed, 9 total
Tests:       1 failed, 5 skipped, 15 passed, 21 total
```

Watch mode

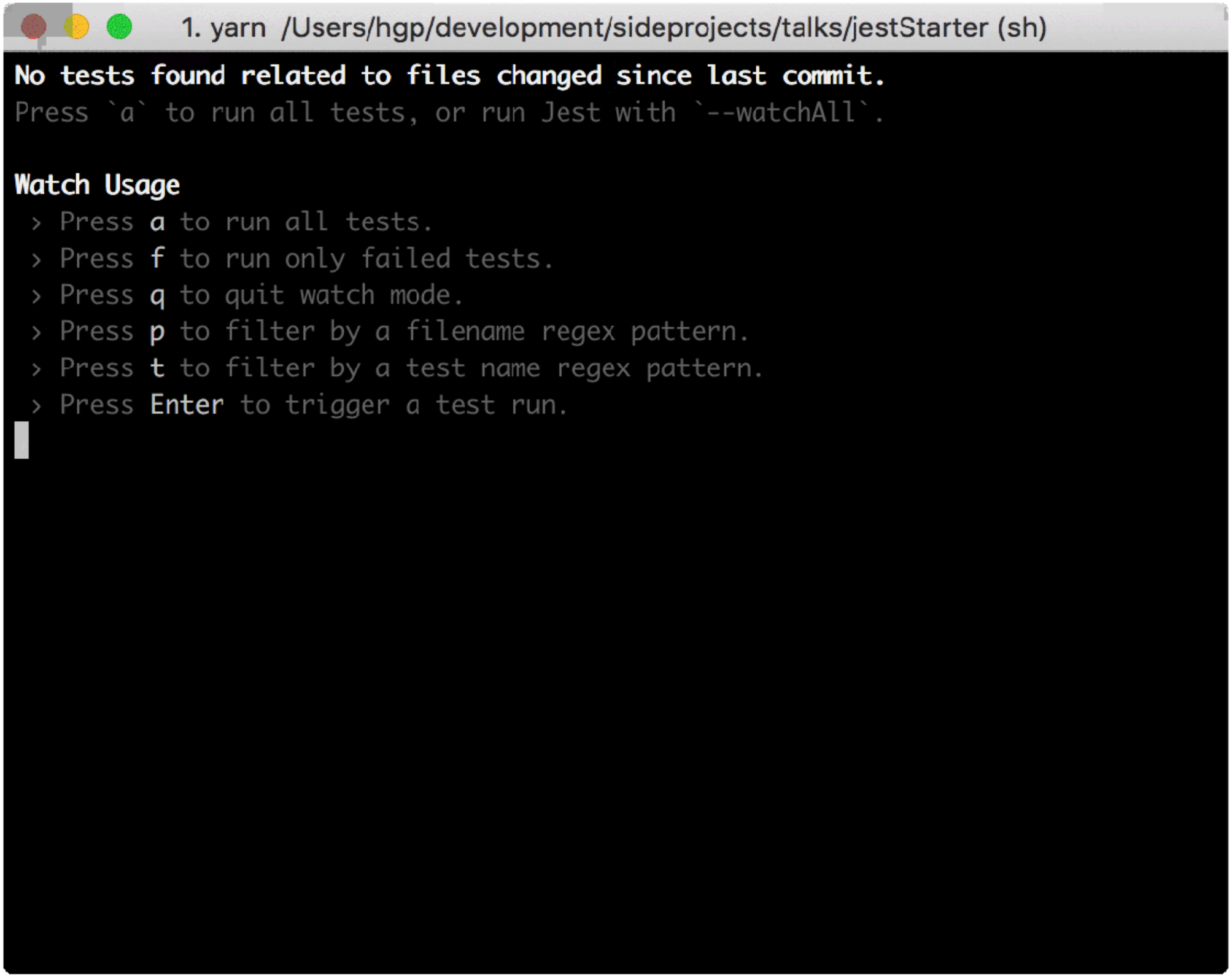
- Looks for changes in the background
- Executes test automatically
- And more!
- Start via:

```
$ jest --watch
```

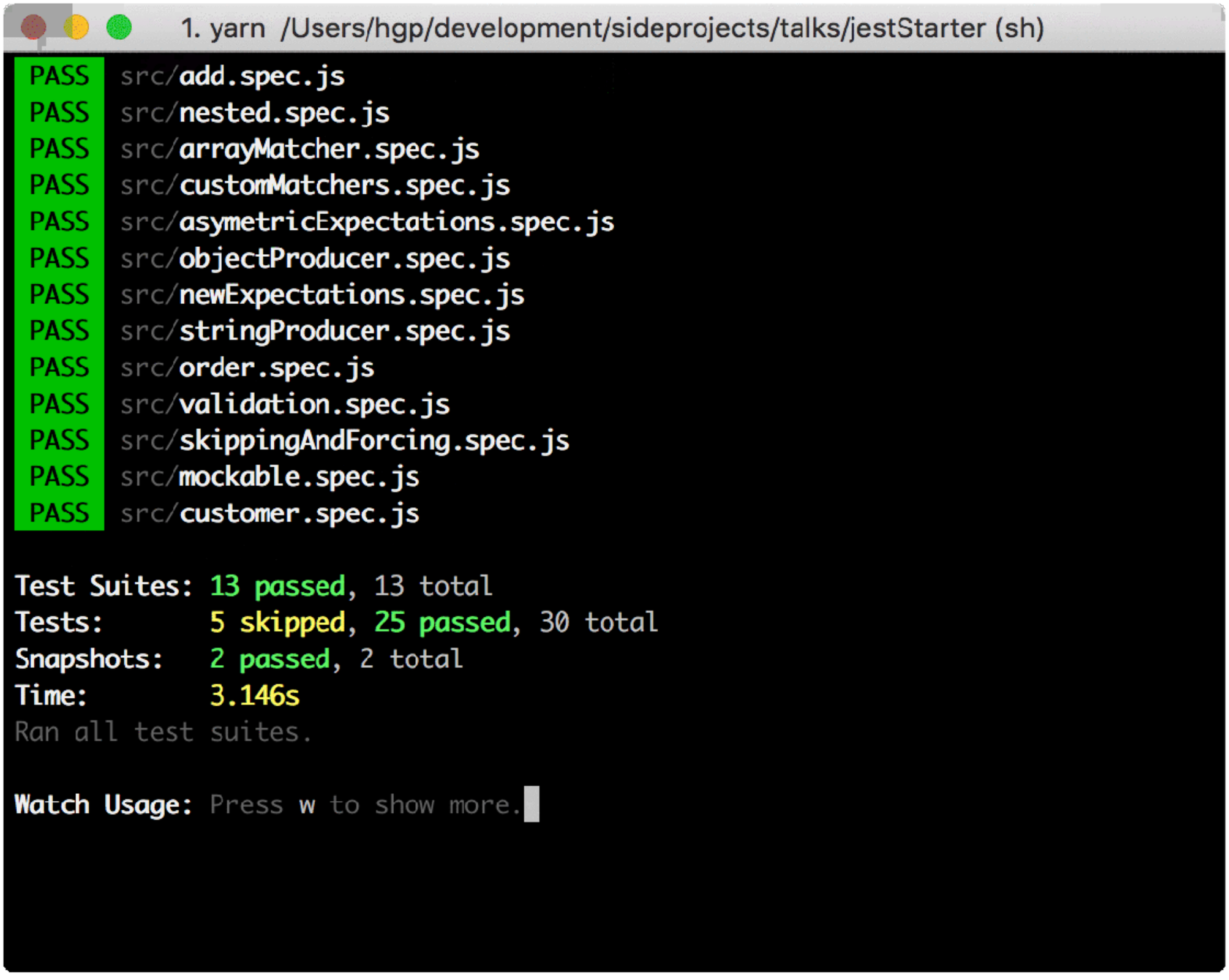
Watch mode Demo



Watch mode Demo



Watch mode Demo

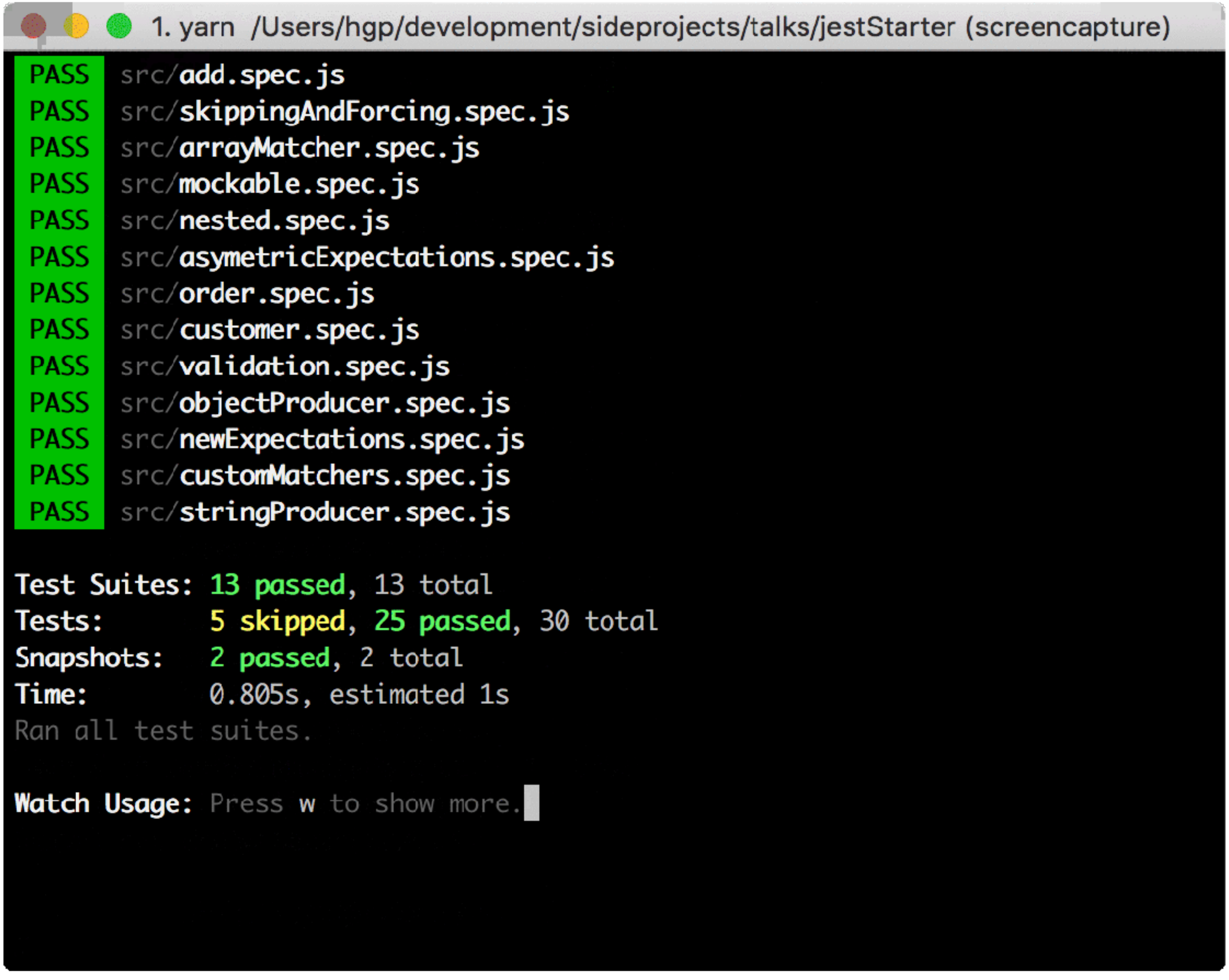


Watch mode Demo

1. yarn /Users/hgp/development/sideprojects/talks/jestStarter (screencapture)

```
    16 | });
    17 |

    at Object.<anonymous> (src/add.spec.js:15:23)

 PASS  src/asymetricExpectations.spec.js
 PASS  src/customMatchers.spec.js
 PASS  src/order.spec.js
 PASS  src/nested.spec.js
 PASS  src/skippingAndForcing.spec.js
 PASS  src/arrayMatcher.spec.js
 PASS  src/newExpectations.spec.js
 PASS  src/customer.spec.js
 PASS  src/stringProducer.spec.js
 PASS  src/objectProducer.spec.js
 PASS  src/mockable.spec.js
 PASS  src/validation.spec.js

Test Suites: 1 failed, 12 passed, 13 total
Tests:       2 failed, 5 skipped, 23 passed, 30 total
Snapshots:   2 passed, 2 total
Time:        0.856s, estimated 1s
Ran all test suites.

Watch Usage: Press w to show more.
```

Watch mode Demo



Best practice: Watch mode on

- You get very fast feedback
- Nice for TDD
- Immersive
  - One less task to do manually
- Make it a habit

Code coverage

- Computes coverage out of the box

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

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

Best practice: Coverage is your friend

- Sane default out of the box
- I use it regularly to see, where some test love may be needed
- Don’t go overboard with the numbers
  - Use it as a guideline
- CI is a good place to make this visible

Even nicer output

- verbose option

```
$ jest --verbose
 PASS  client/components/App.spec.jsx
  App
    ✓ renders (14ms)
    ✓ calling Service correctly (3ms)

 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 async await (1ms)
  httpService getPostsRejecting should
    ✓ reject using done
    ✓ 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.
```

IDE Support

It just works

- **I use it mainly in IntelliJ/Webstorm and it just works**
  - Use a Run Config like any other test

- **Works fine in VSCode**
  - Some more config
    - Not as polished as IntelliJ
    - But fine for me
  - I use the Jest extension

- **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
- Mocking
- Snapshot Tests
- Async Tests
- DOM Testing
- Code coverage

Expectations

- No need for separate Mocha/Jasmine/expect
- All the expectations you need
- Matchers are extendable

https://facebook.github.io/jest/docs/en/expect.html

```
.toBe(value)
.toHaveBeenCalled()
.toBeCloseTo(number, numDigits)
.toBeDefined()
.toBeFalsy()
.toBeGreaterThan(number)
.toBeGreaterThanOrEqual(number)
.toBeLessThan(number)
.toBeLessThanOrEqual(number)
.toBeInstanceOf(Class)
.toBeNull()
.toBeTruthy()
.toBeUndefined()
.toContain(item)
.toContainEqual(item)
.toEqual(value)
.toHaveLength(number)
.toMatch(regexpOrString)
.toMatchObject(object)
...
```

Expectations: Examples

Expecting Objects

```
1 describe('createCustomer should’, () => {
2     it('produce a valid customer',() => {
3         const customer = {name: 'Peter', premium: true};
4         expect(customer).toBeDefined();
5         expect(customer.name).toEqual('Peter');
6         expect(customer.name).toEqual(expect.stringContaining('ete'));
7         expect(customer).toEqual(expect.objectContaining({premium: true}));
8     })
9 });
```

Expecting Arrays

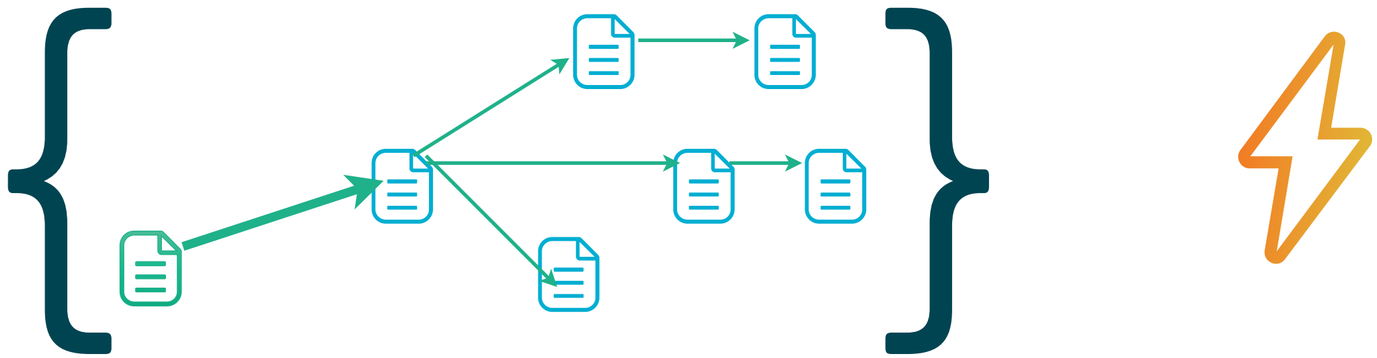
```
1 describe('createCustomers should', () => {
2     it('work with many customer',() => {
3         const customers = [
4             {name: 'Peter', premium: true},
5             {name: 'Max', premium: false},
6             {name: 'Tine', premium: true}
7         ];
8         expect(customers).toContainEqual({name: 'Tine', premium: true});
9     })
1 });
0
```

Best practice: Few expectations

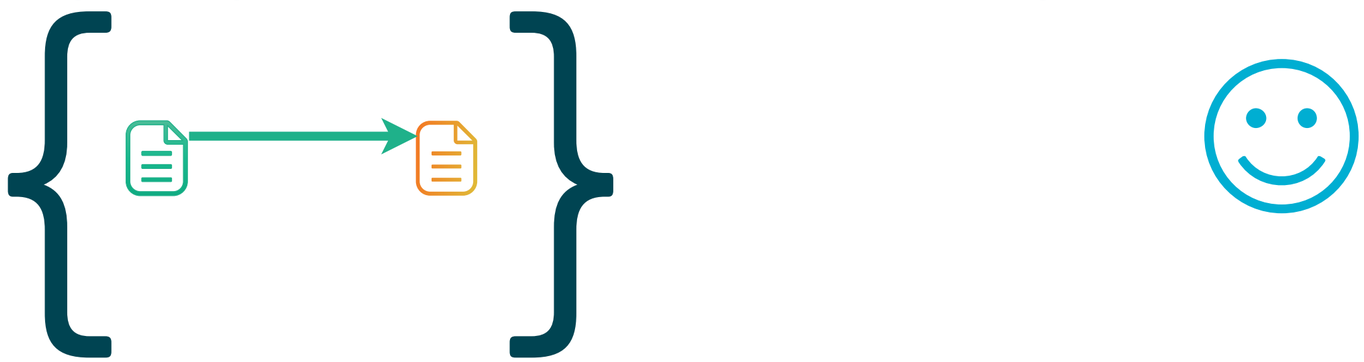
- Don’t go overboard with the expectations in the it blocks
- In doubt write one more it block
- Expectations following a failing expect don’t execute

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

```
1 export function placeOrder(product, qty, seasonId) {
2     if (product.isAvailable(seasonId)) {
3         product.order(qty);
4     }
5 }
```

```
1  describe('placeOrder should', () =>
2      it('order an available product', () => {
3          const product = {
4              isAvailable: jest.fn().mockReturnValue(true),
5              order: jest.fn()
6          };
7
8          placeOrder(product, 2, 'SOMMER');
9
10         expect(product.order).toHaveBeenCalled();
11         expect(product.order).toHaveBeenCalledTimes(1);
12         expect(product.order).lastCalledWith(2);
13         expect(product.order).toHaveBeenCalledWith(2);
14     })
15 );
```

Manual mocks

```
1 import { getOrdersFromRemote } from "./3rdParty/orderRemoteService";
2
3 export let processOrders = somePar => {
4   return getOrdersFromRemote().orders.map(it => it.id);
5 };
```

```
1  import {processOrders} from './orderProcessor';
2
3  jest.mock('./3rdParty/orderRemoteService', () => {
4      const getOrdersFromRemote = () => ({
5          orders: [{id: 1, description: 'TestOrder1'},
6                   {id: 2, description: 'TestOrder2'}]
7      });
8      return {
9          getOrdersFromRemote
10     };
11 });
12
13 describe('orderProcessor should', () => {
14     it('work with returned orders', () => {
15        expect(processOrders()).toEqual([1, 2]);
16     });
17 });
```

Best practice: Mock to reduce scope

- Use mocking if
  - you set things up that don't belong to your test
  - you are only interested in the result (or an error!) of a dependency
- This makes the test more readable and potentially faster
- But beware if you see too much mocking involved
  - Maybe your test can be smaller
  - Maybe you need to restructure your dependencies

Snapshot tests

- **Not (strictly) UI related!**

```
expect(validationMessage('Vorname')).toMatchSnapshot();
```

- Jest stores result of function (i.e any serializable thing) to the fs the first run
  - The stored thing is called **Snapshot**

```
exports[`validationMesssage should produce a valid message 1`] =
`"Feld 'Vorname' ist ein Pflichtfeld."`;
```

- Compares to this in consecutive test runs

```
expect(value).toMatchSnapshot()

 Received value does not match stored snapshot 1.

 - "Feld 'Vorname' ist ein Pflichtfeld."
 + "Das Feld 'Vorname' ist ein Pflichtfeld.“

› 1 snapshot test failed.
```

Best practice: Use Cases for snapshot tests

Test that would require some effort to expect

React components (that should not change)

Things you would not test otherwise

Best practice: Caveats of snapshot tests

Snapshot test may be brittle

You NEED to review your snapshot changes

No jest -u fire and forget

Prefer unit tests

Async tests

- You'll encounter async code quite frequently
- Testing using callback way still works well
  - Known from Jasmine

```
1 it("talk to the backend using done", done => {
2     getCustomers().then(data => {
3       expect(data).toEqual(TEST_RESPONSE);
4       done();
5     });
6 });
```

Async tests

- Return a promise

```
1 it("talk to the backend using promise returns", () => {
2   return getCustomers().then(data => {
3     expect(data).toEqual(TEST_RESPONSE);
4   });
5 });
```

- Use a matcher to resolve the provided promise

```
1 it("talk to the backend using expectations", () => {
2   expect(getCustomers()).resolves.toEqual(TEST_RESPONSE);
3 });
```

- Use async/await

```
1 it("talk to the backend using async await", async () => {
2   const customers = await getCustomers();
3   expect(customers).toEqual(TEST_RESPONSE);
4 });
```

DOM testing

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

- In a test you can setup a mock structure of your sites markup

```
1   const setupDOM = () => {
2       document.body.innerHTML =
3           '<div>' +
4           '  <div id="myText">Lorem</div>' +
5         '</div>';
6   };
```

- Assert in the DOM

```
1 it('show text on page', () => {
2     const text = document.querySelector('#myText').innerHTML;
3     expect(text).toEqual('Lorem');
4 });
```

- Useful when in conjunction with e.g. JQuery

Best practice: Keep away from DOM

- As long as you can
- Tests will be slow and brittle
- If you must expect sth in the DOM, keep it brief

DOM testing (outlook)

Test the rendering of your React application:

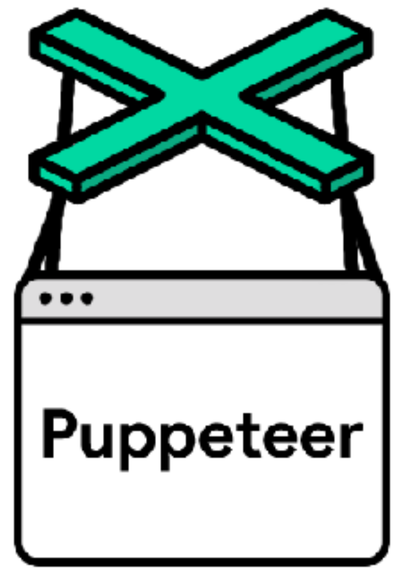
Enzyme (http://airbnb.io/enzyme/)

Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.

Browser testing using Jest:

Puppeteer (https://github.com/GoogleChrome/puppeteer) and its **Jest integration** (https://facebook.github.io/jest/docs/en/puppeteer.html)

Puppeteer is a Node library which provides a high-level API to control headless Chrome or Chromium over the DevTools Protocol.



What's in the box

- Expectations
- Mocking
- Snapshot Tests
- Async Tests
- DOM Testing
- Code Coverage
- And more

Coming to an end

Some closing words

What we had to do

- Install dependencies
- package.json
- babel

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

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

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

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

Try Jest yourself

Easy to use and setup

Powerful features set yet familiar

Polished product (even the CLI)

Some links

Tryout Jest

http://jestjs.io/

Release Blog

http://jestjs.io/blog/

Curated Links

github.com/jest-community/awesome-jest

Sample Code

github.com/holgergp/jestStarter


START
TESTING
YOUR
JS
TODAY


Thanks!

Questions?

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

Picture Links

- https://unsplash.com/photos/xA5QE8Gtaak
- https://unsplash.com/photos/Wiu3w-99tNg
- https://unsplash.com/photos/FQjmQgSoRyQ
- https://unsplash.com/photos/h1v8lkfDUu4
- https://unsplash.com/photos/58t0B36ZTnw
- https://unsplash.com/photos/cEUl-0DSM9s
- https://unsplash.com/photos/9HI8UJMSdZA
- https://unsplash.com/photos/xhWMi9wdQaE
- https://unsplash.com/photos/R6xx6fnvPT8
- https://unsplash.com/photos/Q6jX7BbPE38
- https://unsplash.com/photos/8xAA0f9yQnE
- https://unsplash.com/photos/0YbeoQ0X89k
- https://unsplash.com/photos/D56TpVU8zng
- https://unsplash.com/photos/TyQ-0lPp6e4
- https://unsplash.com/photos/gdBXlL053N4
- https://unsplash.com/photos/uAFjFsMS3YY