

TESTING JAVASCRIPT

BUILDING JAVASCRIPT APPLICATIONS YOU WON'T HATE

1/45 - TESTING JAVASCRIPT

Called Testing JS but really
about embracing the front end



Ölbaum
@oscherler



JavaScript makes me want to flip the table and say “Fuck this shit”, but I can never be sure what “this” refers to.

7:03 AM · Oct 30, 2015 · [Twitterrific for Mac](#)

1.2K Retweets 1.3K Likes



Classic JS meme about "this"
This tweet could have ended half way through and still been relevant to my early JS days
Is the language bad or do I just not have the tools and experience to write it?



3/45 - TESTING JAVASCRIPT

Early JS apps always seemed to degrade into a dumpster fire
Over time I've learned to love building javascript frontends.
Big push to embracing the backend but in this market no escaping JS

WHY DO WE TEST?

4/45 - TESTING JAVASCRIPT

The great thing about testing is everyone has an opinion about how it should be done.

There are people who've been writing tests for code longer than I've been alive so won't lecture on that
Talk about some tool I use and when and what's worked for me (some opinion thrown in)

Want people to keep this in mind while we go over the tools

People should be asking "Will this tool make my life easier or not?"

CORRECTNESS

5/45 - TESTING JAVASCRIPT

Sometimes not obvious if testing manually. Is someone really going to check GST value is correct during every QA run?

REFACTORING

6/45 - TESTING JAVASCRIPT

1. making changes and re-running the test suite is less stressful.
2. Extracting things out to their own components as they're reused

SCALE

7/45 - TESTING JAVASCRIPT

1. App complexity e.g. feature set
2. developers on the team
3. Summed up as better dev experience



Jonathan H. Wage

@jwage



5 months ago I joined a team which had an app with 500k lines, 0 tests and a monthly manual ftp deploy. Today we have automated deploys and 1588 tests and 23761 assertions. We've deployed almost 100 times since then. Still a long road ahead but I know we are on the right path.

♡ 397 12:43 AM - Oct 6, 2019



💬 46 people are talking about this



IMPROVE DEVELOPER EXPERIENCE

1. I want to have an easy life. Already writing javascript, do we want it to be any more stressful?

2. Last place not many tests. Stressful to deploy. Shit broke. No deploy Fridays etc

3. We got it to a point we were confidently Shipping 10 times a day. Even 5pm Friday

WHAT DO WE TEST?

9/45 - TESTING JAVASCRIPT

Opinion territory. Been covered to death

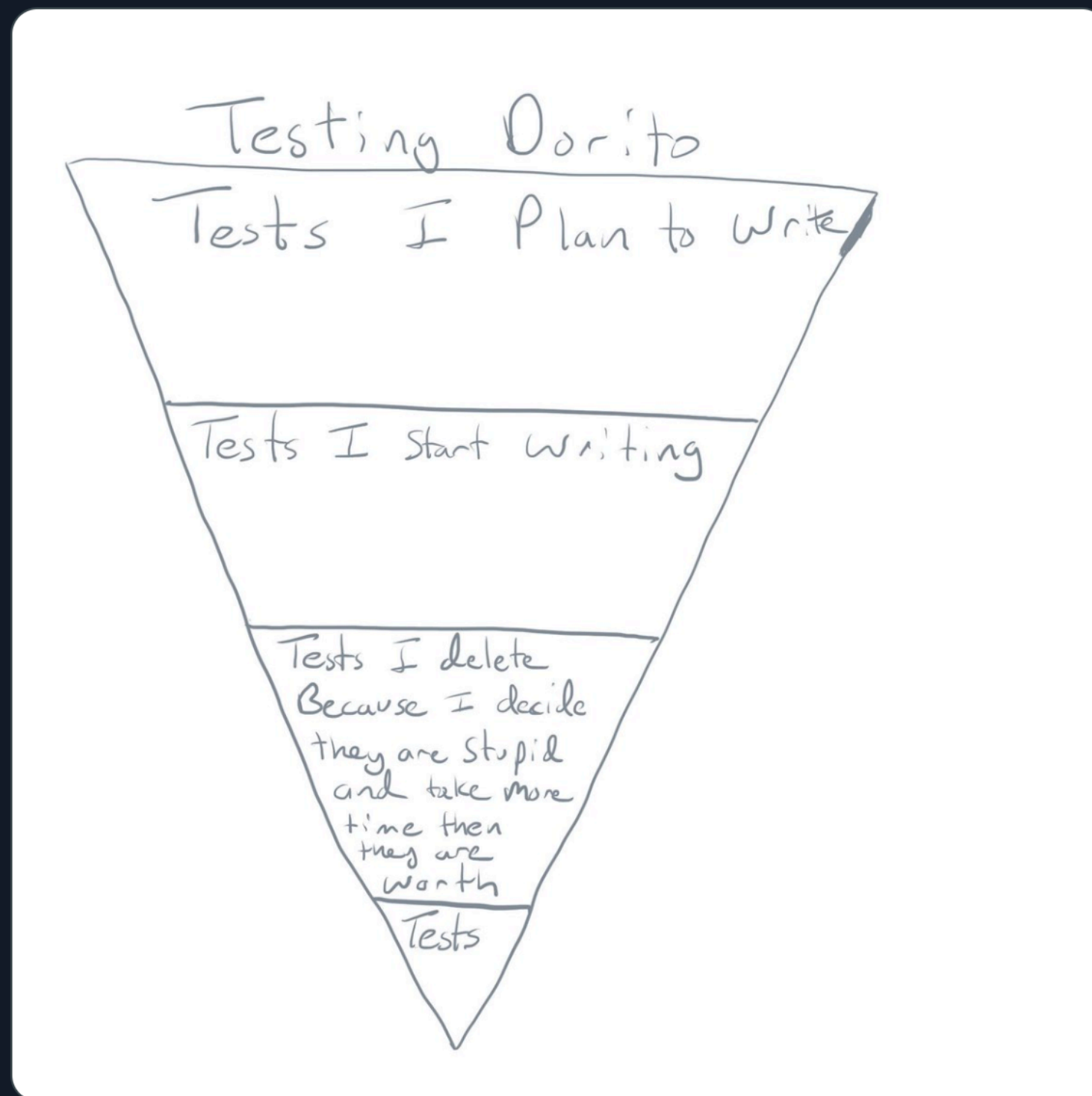
Everyone works on different things. Quickly skim some classic takes and then move on

I want to go over tools and you can figure out which ones work for you



Tim Myers
@denvercoder

Replying to @kentcdodds @Cypress_io and 3 others



4:20 PM · Feb 6, 2018 · Twitter for iPad

love this tweet. Not only true but highlights cost of tests. If tests free people wouldn't be coming up here on stage or writing books telling you to test. So opinion's, lets have them



Guillermo Rauch

@rauchg

Following



Write tests. Not too many. Mostly integration.

8:43 AM - 10 Dec 2016 from San Francisco, CA

111 Retweets 357 Likes



16



111



357



11/45 - TESTING JAVASCRIPT

Guillermo isn't just some random person
Creator of ZEIT, creator of OS like socket.io and mongoose
Probably knows a thing or two about (some) software



INTEGRATION TESTS ARE A SCAM

– J.B. RAINSBERGER

Great talk and articles (albeit a little circular). Go watch/read them
Test have cost as well as value. We want valuable and cheap tests.
I tend to agree with this for front end testing. You'll see why soon.
Even if you disagree that's fine. I'm covering a wide range of tools.

TESTING TOOLKIT

13/45 - TESTING JAVASCRIPT

1. Go through each one pretty quickly and high level.
2. Not going to be too many code examples because all of the tools have great documentation and resources
3. This is the worst possible medium for teaching you how to implement
4. Just sit back and relax and just ask yourself at each tool "Will this make my life easier"
5. If it does go look at the docs and find resources on them. They're all great!

STATIC ANALYSIS

ESLINT & TYPESCRIPT

14/45 - TESTING JAVASCRIPT

1. Linting (more than just indentation)
2. Spend more time reading code than writing it
3. Reduce cognitive overhead - Matt Stauffer's point last year on code style.
5. Instant feedback on basic errors
6. Missing imports, unused vars, unreachable statements, missing assertions, spelling errors, ensure test file etc
7. Teaches you how to write JS "better"
8. Custom ES lint rules (e.g test file exists)

I WRITE TESTS ANYWAY, SO I DON'T NEED A TYPE CHECKER
- SOMEONE WHO IS WRONG

15/45 - TESTING JAVASCRIPT

1. Gary Bernhardt (Destroy All Software) great talk on this
2. Gary makes two main points

TESTS ARE EXAMPLES

CORRECTNESS IS HARD TO PROVE FROM EXAMPLES

16/45 - TESTING JAVASCRIPT

1. If this wasn't true the word "edge case" wouldn't exist

TYPES DEFINE CATEGORIES

CATEGORIES CANNOT PROVE CORRECTNESS

17/45 - TESTING JAVASCRIPT

1. Tests catch type errors at compile time
2. Will fail at first type error on runtime
(meaning no unexpected follow on effects)
3. Low effort, Moderate Value, Instant Feedback
4. Just like tests they're a form of documentation. Arguable easier to read
5. If you don't like types then don't use them. If you do they are here.

UNIT TESTING

JEST

18/45 - TESTING JAVASCRIPT

1. easy and fast
2. Fail for one reason and one reason only (easier to debug)
3. Encourage pure functions
4. Functional composition tends to scale better *for me*
5. My favourite tests

INTEGRATION TESTING

JEST

19/45 - TESTING JAVASCRIPT

1. Testing how Vue or React components interact
2. Can hook into Vue/React easily and test child components etc
3. Honestly I write basically 0 of these on the F.E.
4. I structure my code in a way that tries to maximise isolation between components (state management)

VUEX / REDUX

20/45 - TESTING JAVASCRIPT

1. Makes app really easy to unit test. Set some state, perform an action (?), make assertions
2. All your side effects now pushed to the edge
3. Avoid Prop drilling and Event Buses
4. Much fewer code paths (don't have to test AB AC AD BC BD CD etc. Just A B C D)



THERE IS ANOTHER

1. There's one more kind of test that Jest allows
2. SNAPSHOT TESTING (opinion time)
3. I don't use it ever - maybe one day I will but you should know it's there
4. outputs to seperate directory (*people won't check it*)
5. By definition you can't do *TDD*
6. useless if you don't validate original snapshot**
7. Merge Conflicts on Snapshots Suck!



1. Why is it a thing if its so bad
2. People love it because easy and fast code coverage
3. If you have an app with no tests, and want to refactor and change nothing it could be useful
4. Try snapshot artifacts like code --> Reviewed as part of code review process
5. eslint no large snapshots

LET'S WRITE A JEST TEST

TYPESCRIPT FTW

```
interface Item {  
  name: string  
  sku: string  
  price: number  
}
```

```
interface CartItem extends Item {  
  quantity: number  
}
```

```
type Cart = Array<CartItem>
```

```
export default function addItemToCart(item: Item, cart: Cart) {  
  const [matchedItem]: Array<CartItem> = cart.filter((cartItem: CartItem) => cartItem.sku === item.sku);  
  
  if (matchedItem) {  
    matchedItem.quantity += 1;  
  } else {  
    cart.push({...item, quantity: 1});  
  }  
  
}
```

```
export default function addItemToCart(item: Item, cart: Cart) {  
  const [matchedItem]: Array<CartItem> = cart.filter((cartItem: CartItem) => cartItem.sku === item.sku);  
  
  if (matchedItem) {  
    matchedItem.quantity += 1;  
  } else {  
    cart.push({...item, quantity: 1});  
  }  
  
}
```

```
export default function addItemToCart(item: Item, cart: Cart) {
  const [matchedItem]: Array<CartItem> = cart.filter((cartItem: CartItem) => cartItem.sku === item.sku);

  if (matchedItem) {
    matchedItem.quantity += 1;
  } else {
    cart.push({...item, quantity: 1});
  }
}
```

```
test('adding an item to an empty cart makes the cart length equal to 1', () => {  
  // Setup  
  const cart: Cart = [];  
  
  const item: Item = {  
    name: 'Some really good item',  
    sku: 'SKU_FOO_BAR_BAZ_123',  
    price: 2999,  
  };  
  
  // Act  
  addItemToCart(item, cart);  
  
  // Assert  
  expect(cart).toHaveLength(1);  
});
```



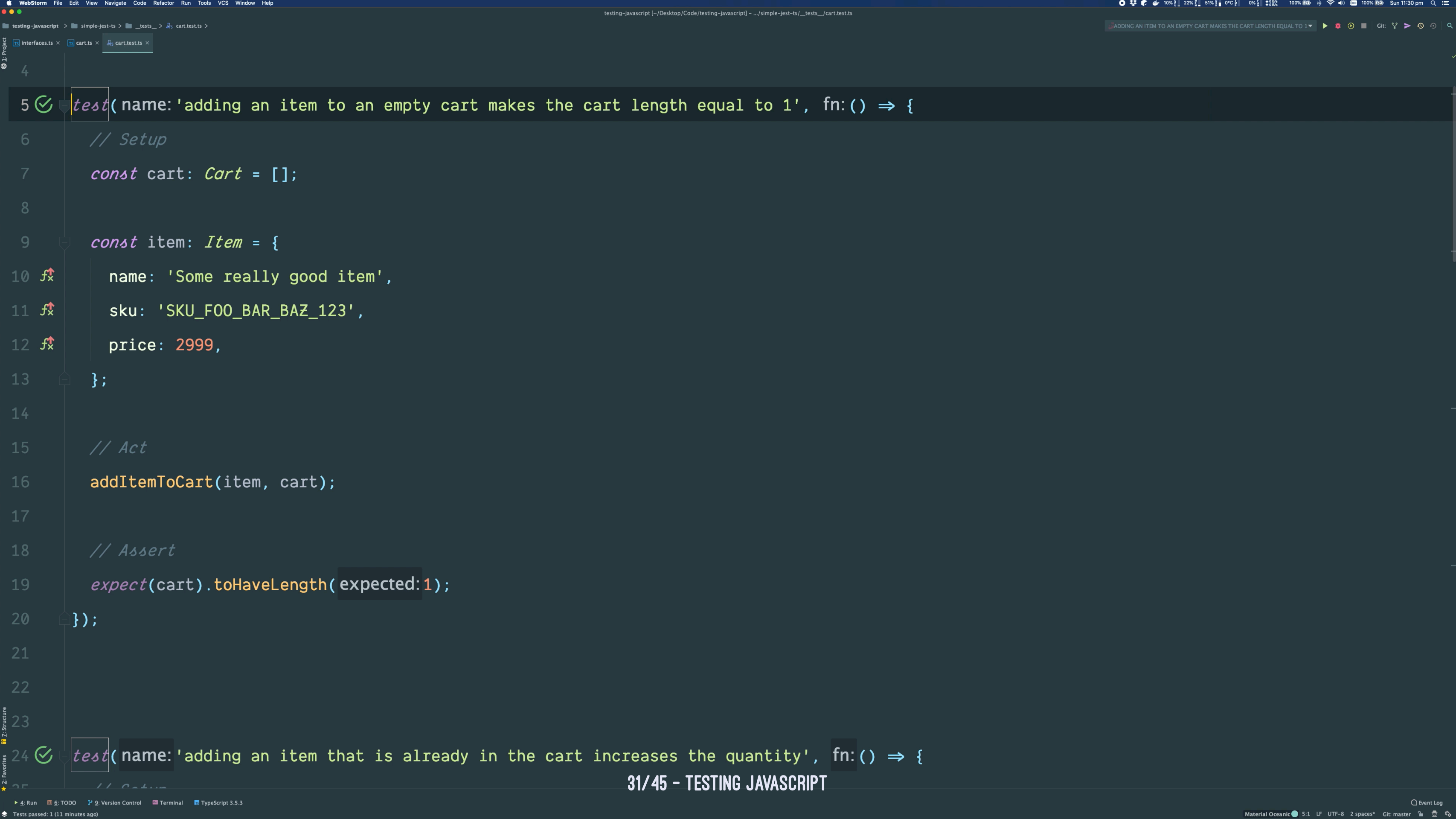
```
test('adding an item to an empty cart makes the cart length equal to 1', () => {
  // Setup
  const cart: Cart = [];

  const item: Item = {
    name: 'Some really good item',
    sku: 'SKU_FOO_BAR_BAZ_123',
    price: 2999,
  };

  // Act
  addItemToCart(item, cart);

  // Assert
  expect(cart).toHaveLength(1);
});
```

```
test('adding an item to an empty cart makes the cart length equal to 1', () => {  
  // Setup  
  const cart: Cart = [];  
  
  const item: Item = {  
    name: 'Some really good item',  
    sku: 'SKU_FOO_BAR_BAZ_123',  
    price: 2999,  
  };  
  
  // Act  
  addItemToCart(item, cart);  
  
  // Assert  
  expect(cart).toHaveLength(1);  
});
```



The screenshot shows a code editor with a dark theme. The file explorer on the left shows a project structure with 'testing-javascript' as the root, containing 'simple-jest-ts' and 'tests'. The 'tests' folder contains 'cart.test.ts'. The code in 'cart.test.ts' is as follows:

```
4
5 test(name: 'adding an item to an empty cart makes the cart length equal to 1', fn: () => {
6     // Setup
7     const cart: Cart = [];
8
9     const item: Item = {
10         name: 'Some really good item',
11         sku: 'SKU_FOO_BAR_BAZ_123',
12         price: 2999,
13     };
14
15     // Act
16     addItemToCart(item, cart);
17
18     // Assert
19     expect(cart).toHaveLength(expected: 1);
20 });
21
22
23
24 test(name: 'adding an item that is already in the cart increases the quantity', fn: () => {
25     // Setup
```

The status bar at the bottom shows '31/45 - TESTING JAVASCRIPT' and 'Tests passed: 1 (11 minutes ago)'.

1. My favourite feature in jest is code coverage
2. Ironically this is more useful for integration tests. But I use jest a lot for node so 🙄
3. If you use JetBrains (Webstorm/PHPStorm) this is what it looks like
4. I'm sure it's possible with VSCode and Vim, just don't ask me how

```
23
24 test(name: 'adding an item that is already in the cart increases the quantity', fn: () => {
25     // Setup
26     const item: Item = {
27         name: 'Some really good item',
28         sku: 'SKU_FOO_BAR_BAZ_123',
29         price: 2999,
30     };
31
32     const cart: Cart = [cartItem(item, quantity: 1)];
33
34     // Act
35     addItemToCart(item, cart);
36
37     // Assert
38     expect(cart[0].quantity).toBe(expected: 2);
39 });
40
41
42
```

32/45 - TESTING JAVASCRIPT

1. Here we write another test.

Test other code path

MUTATION TESTING

STRYKER

33/45 - TESTING JAVASCRIPT

1. Mutation testing is cool
2. It randomly mutates your code and re-runs your test suite on the code it has changed
3. Why would we want to do this?



1. Tests the **quality of your tests, rather than your code
2. If you were paying attention my unit tests examples weren't great. 100% coverage but some problems.
2. Most codebases have tests that will give false positives
3. These are kind of slow. No need in pipeline. Only run on unit tests

→ simple-jest-ts git:(master)

35/45 - TESTING JAVASCRIPT

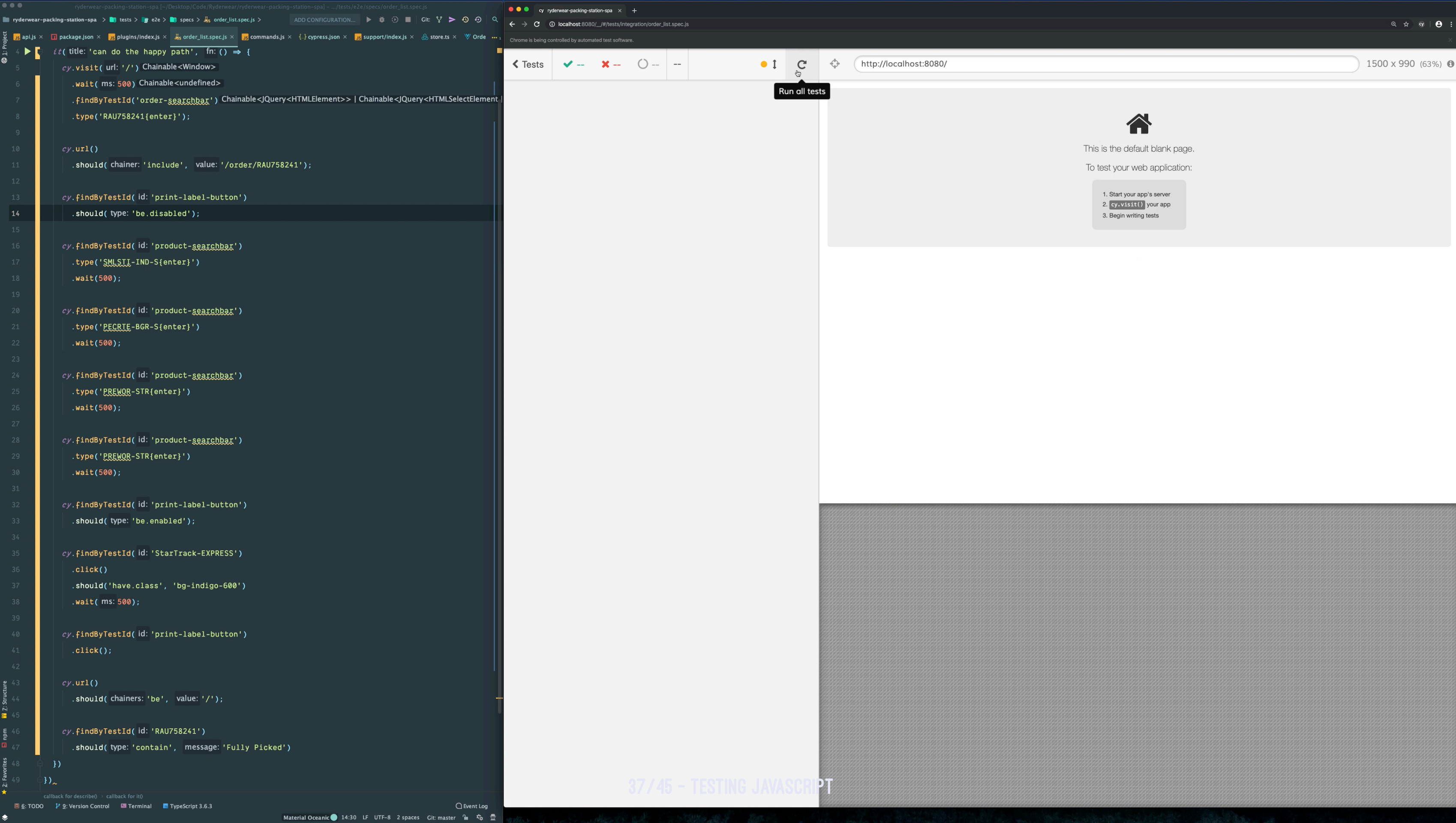
1. First thing it will run test suite - no point running if tests already fail
2. 2 issues with our unit tests.
3. Start with second case. Tests will pass if filter always returns true (we only asserted on quantity of first item in cart
4. We fix that with another test and rerun
5. Now because our lazy test of asserting cart length allows mutants because we can push anything to array
6. So now we assert on cart content
7. We see Stryker won't run if unit tests don't pass
8. Fix unit tests and rerun and we get 100% mutants killed

E2E TESTING

CYPRESS.IO

36/45 - TESTING JAVASCRIPT

1. E2E generally Slower to write
2. Generally not a fan but cypress is actually amazing
3. Correctness for important workflows only. Brittle to maintain
4. Don't bother asserting cart totals or anything stupid. Thats what Unit Tests do
5. Cool video recordings from the test



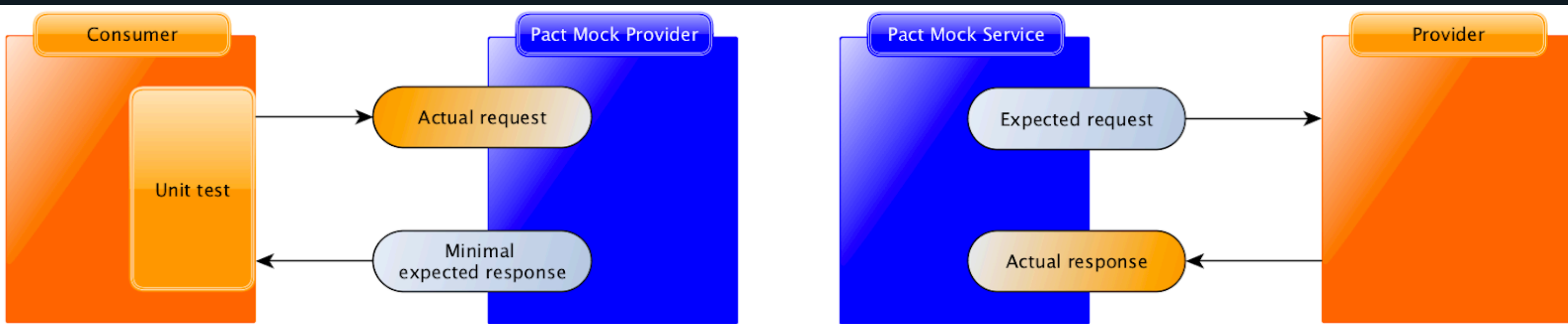
1. Example from small internal app I built for a client
2. Nice fluent API for clicking/typing/find by element
3. Assertions on classes and content etc
4. Really nice to write and very visual

CONTRACT TESTING

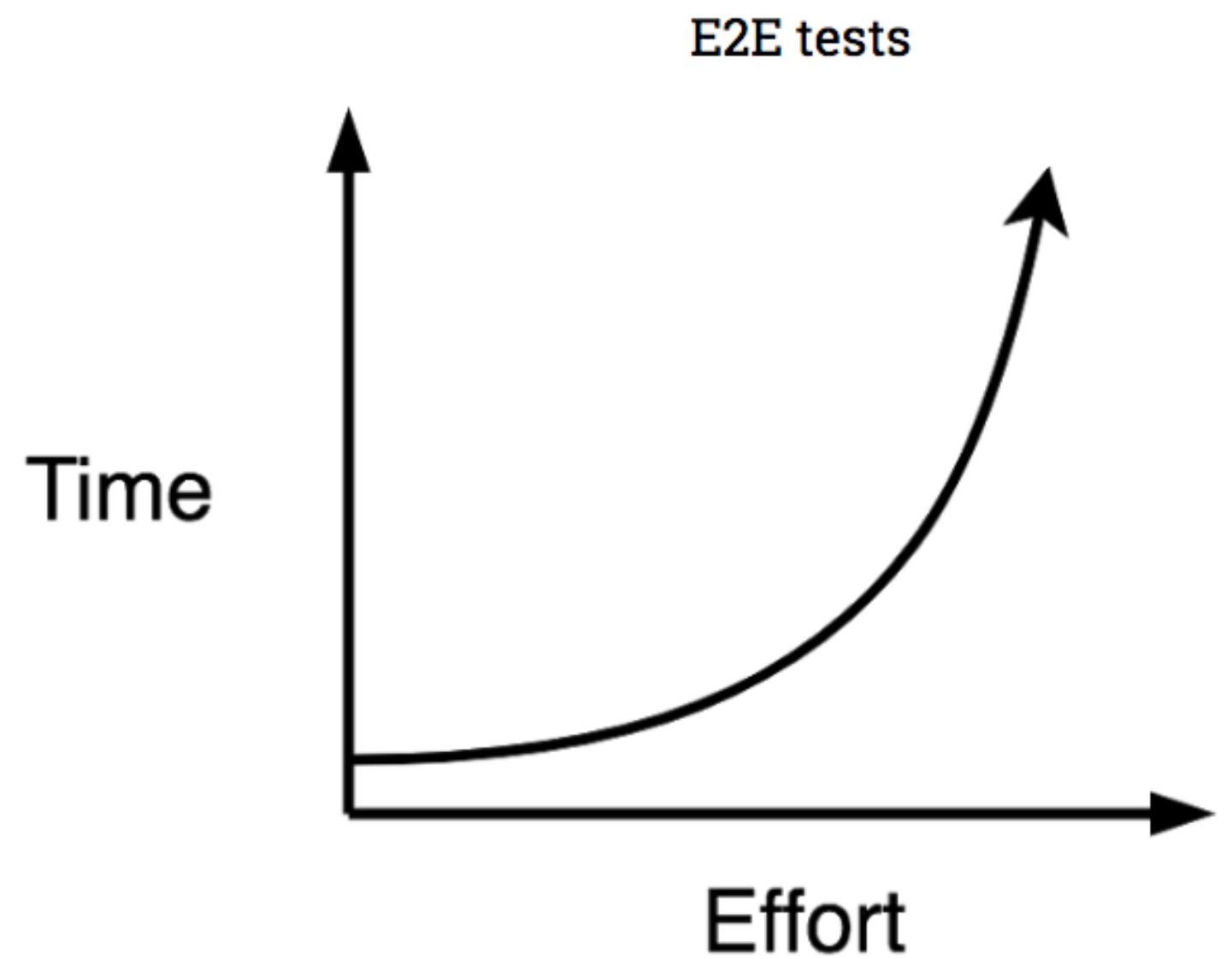
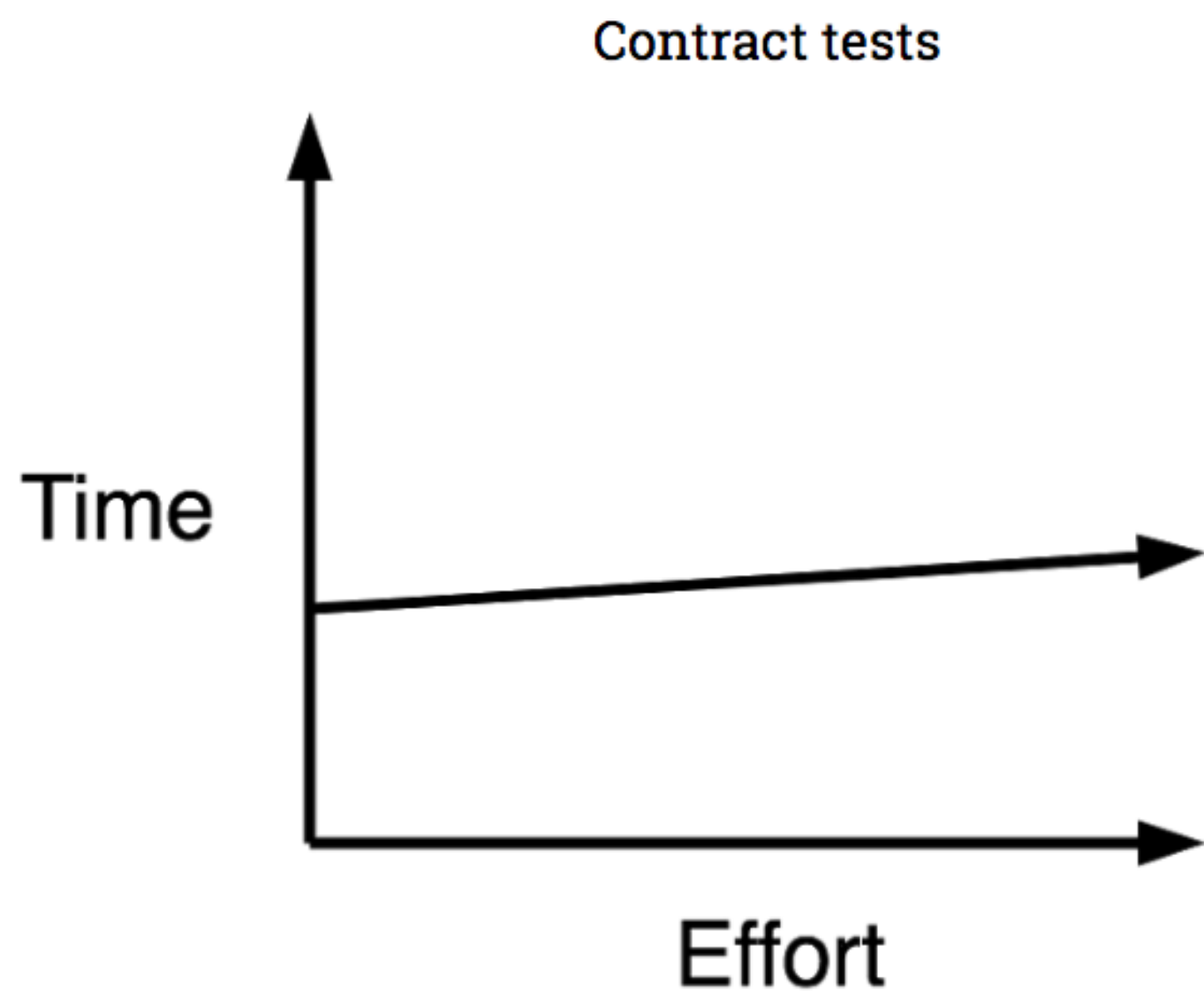
PACT.IO

38/45 - TESTING JAVASCRIPT

1. Not super relevant if using inertia or are passing data from blade into component props
2. Once you fully embrace independent client/server apps it becomes valuable
3. **consumer driven** tests that make sure client and API are in sync



1. Throws a server between client and API
2. Run test suite against pact server, caches requests
3. Next time API has a change and goes through CI/CD it sends those cached requests to the API
4. If they don't get back the responses expected it will **FAIL THE API BUILD!**
5. Amazing right? Anything that the FE doesn't care about can change to whatever it likes
6. Stops other teams/developers breaking your code
7. Also on the downstream end it caches the responses server sends back and makes sure your front end can handle those message formats too



1. Contract tests keep the effort low of maintaining segregated backend/frontend
2. We obviously need to control API and consumer
3. Can't do if you have other consumers of API (I.e. you expose a public version to customers too JSON Schema Validation)
4. Allows us to easily evolve codebase knowing Pact will guarantee contracts are met without having to do strict versioning
5. Find before deploy if things will break (no need for slow E2E tests)
6. Contracts managed by Pact not any individual repo.



41/45 - TESTING JAVASCRIPT

1. Important to remember to focus on the **messages** rather than the behaviour
2. It can be tempting to use contract tests to write general functional tests for the provider
3. Public APIs
4. Passthrough API's (*queues*) always going to 2xx response


```
Given "there is no user called Mary"  
When "creating a user with username Mary"  
    POST /users { "username": "mary", "email": "...", ... }  
Then  
    Expected Response is 200 OK
```

42/45 - TESTING JAVASCRIPT

Sticking to happy-paths is a risk of missing different response codes and potentially having the consumer misunderstand the way the provider behaves

```
Given "there is already a user called Mary"  
When "creating a user with username Mary"  
  POST /users { "username": "mary", "email": "...", ... }  
Then  
  Expected Response is 409 Conflict
```

43/45 - TESTING JAVASCRIPT

So far so good, we're covering a new behaviour, with a different response code.

This is where we get on the slippery slope... it's very tempting to now add scenarios to our contract, something like:

```
When "creating a user with a blank username"
  POST /users { "username": "", "email": "...", ... }
Then
  Expected Response is 400 Bad Request
  Expected Response body is { "error": "username cannot be blank" }
```

```
When "creating a user with a username with 21 characters"
  POST /users { "username": "thisisalooongusername", "email": "...", ... }
Then
  Expected Response is 400 Bad Request
  Expected Response body is { "error": "username cannot ...." }
```

44/45 - TESTING JAVASCRIPT

We've gone past the contract testing at this point, we're actually testing that the User Service implements the validation rules correctly: this is functional testing, and it should be covered by the User Service in its own codebase.

What is the harm in this... more testing is good, right? These scenarios are going too far and create an unnecessarily tight contract - what if the User Service Team decides that actually 21 characters is fine?

IF I HAVE SEEN FURTHER,
IT IS BY STANDING
**ON THE SHOULDERS
OF GIANTS.**

- ISAAC NEWTON



If I have seen *at all* it's because
of this great community
Its treated me really well and
I'm glad we have events like
this to help it grow