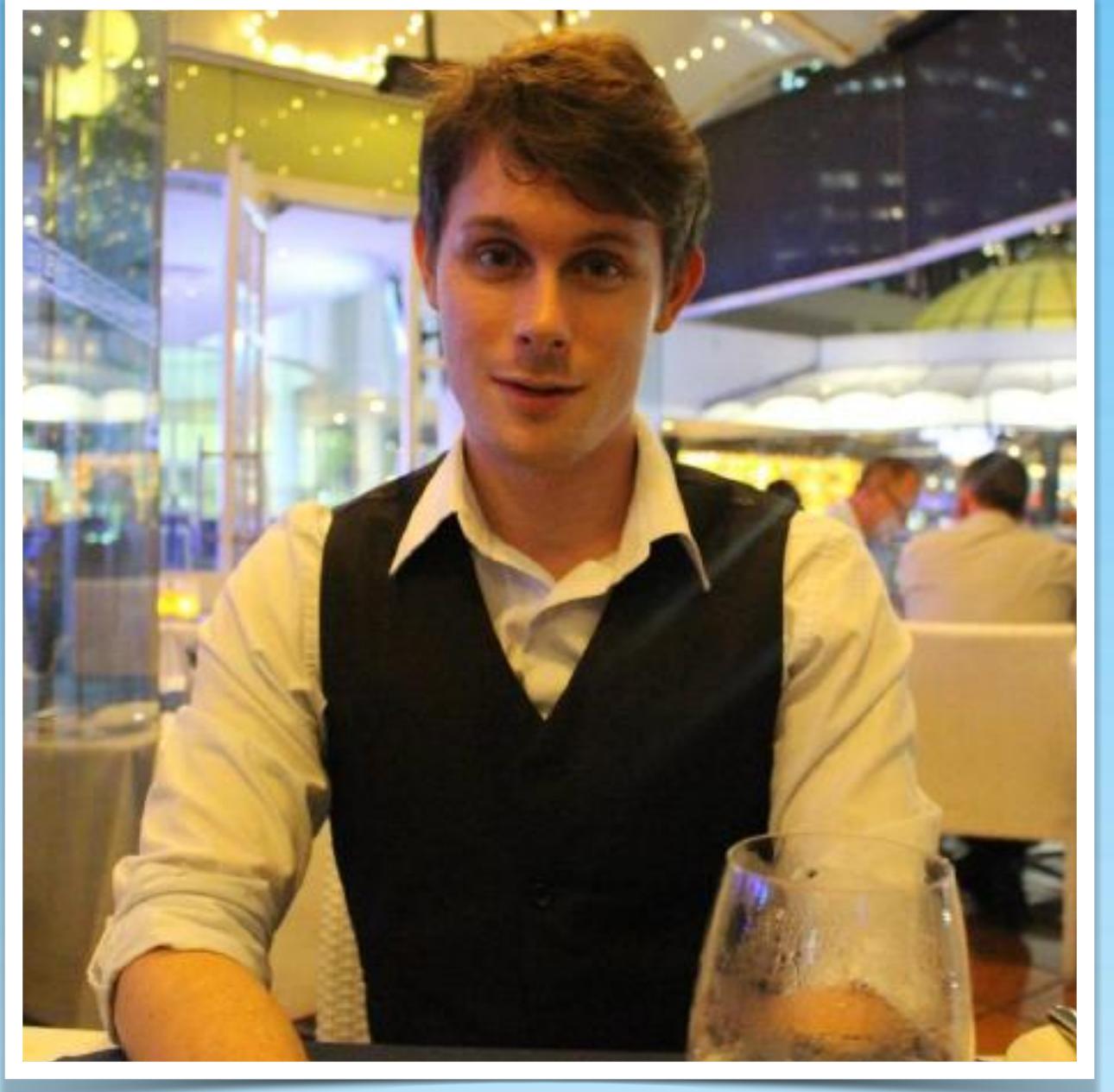


The State of Vue Testing

**Presented by Lachlan Miller
Vue Toronto 2020**

Who am I

- Hi 🙌 I'm Lachlan
- Contributing to **Vue Test Utils** since 2017
- Maintaining since 2019, along with **Vue Jest**
- Author [Vue Testing Handbook](#)
- Twitter: @Lachlan1990
- GitHub: lmiller1990



Lachlan Miller | State of Vue Testing
<https://lachlan-miller.me/vue-toronto-2020>

Agenda

1. State of Vue Test Utils
 1. A new code base
 2. What's new? What's changed?
2. A new philosophy
3. Resources

Get the slides: <https://lachlan-miller.me/vue-toronto-2020>

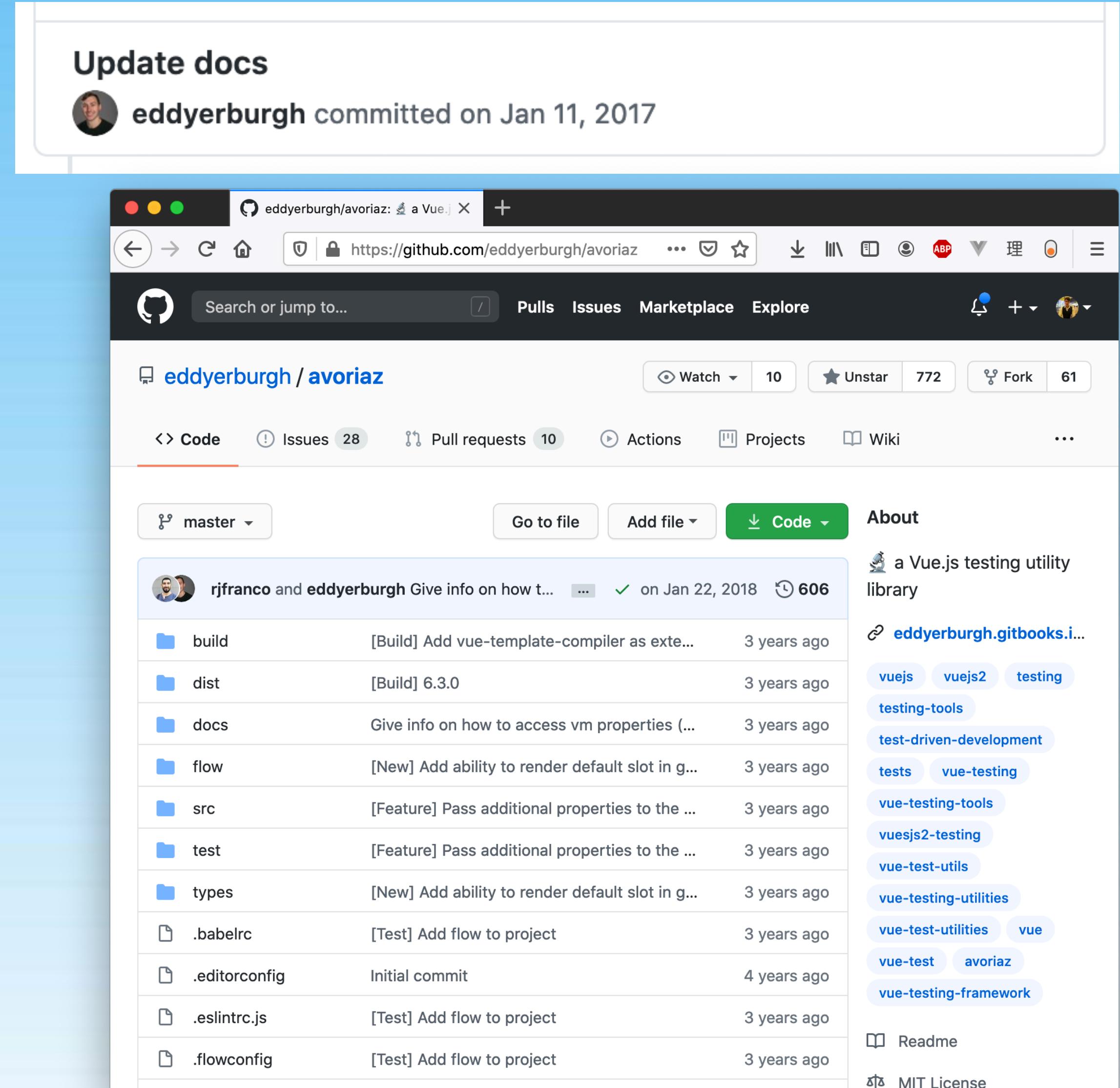
The journey so far...

- Simple "component centric" API for testing Vue components
- Mount components
- Get a "wrapper" with convenient method for testing

```
7  test('renders a message', () => {  
8    const wrapper = mount(Hello, {  
9      props: {  
10        msg: 'Hello'  
11      }  
12    })  
13    wrapper.find('button').trigger('click')  
14    expect(wrapper.html()).toContain('Hello')  
15  })
```

The journey so far...

- Jan 11, 2017 - "Enzyme for Vue" was born as "Avoriaz"
- A way to render and manipulate a component
- With "mounting options"



The journey so far...

- Mid 2017 - Avoriaz used a base for Vue Test Utils
- We have a nice little library that seems to be working great - what could go wrong?

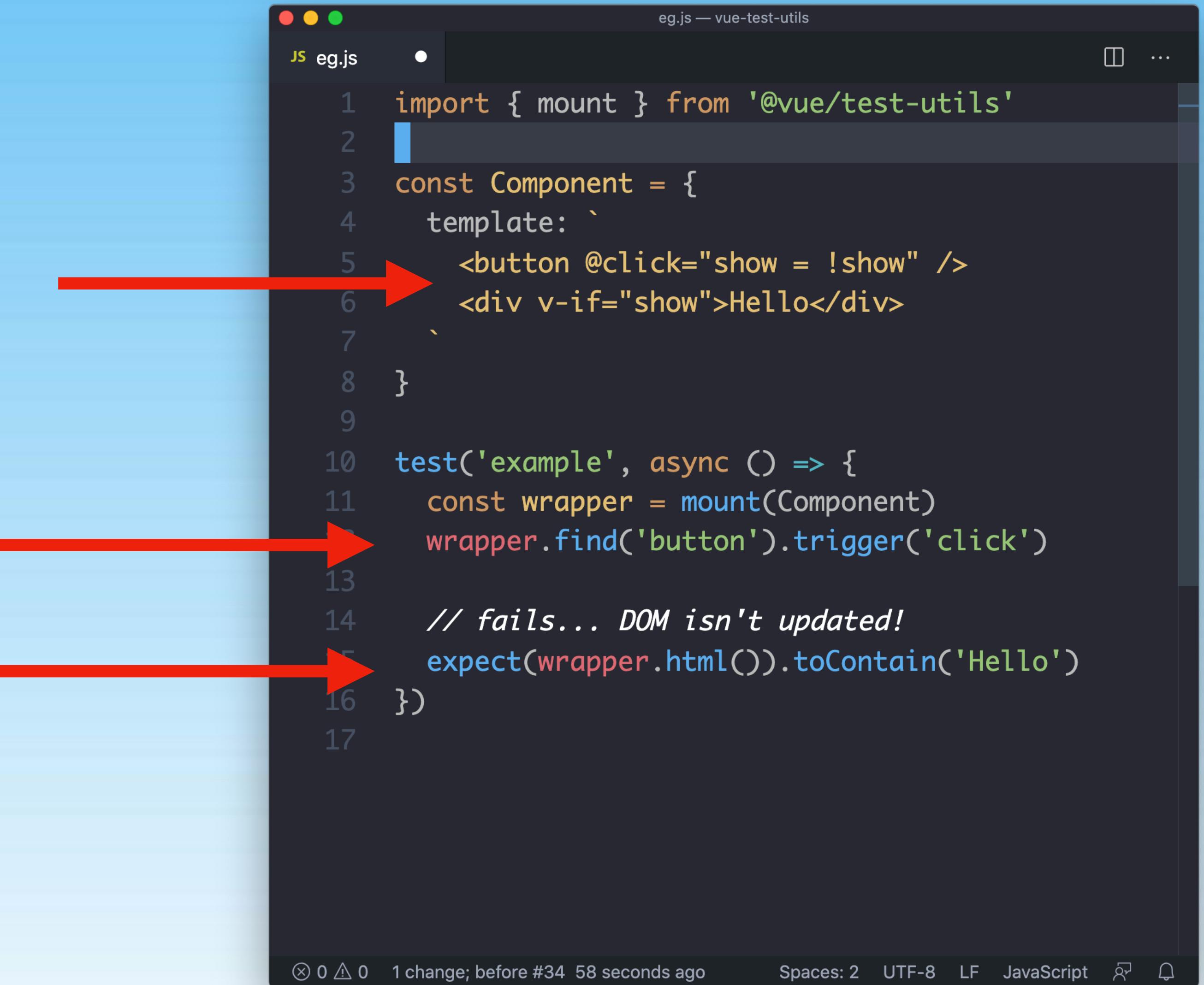
```
commit 309d802b60673310cb2a9461c902255d6821335c
Author: Edd Yerburgh <edward.yerburgh@gmail.com>
Date:   Wed May 31 19:54:49 2017 +0100

Initial commit 🚀
```

```
7  test('renders a message', () => {
8    const wrapper = mount(Hello, {
9      props: {
10        msg: 'Hello'
11      }
12    })
13    wrapper.find('button').trigger('click')
14    expect(wrapper.html()).toContain('Hello')
15  })
```

Vue renders async; runners are not

- This test fails! 🤔
- Test runner will execute the assertion before Vue has re-rendered the DOM
- Unintuitive!



```
JS eg.js • eg.js — vue-test-utils
1 import { mount } from '@vue/test-utils'
2
3 const Component = {
4   template: `
5     <button @click="show = !show" />
6     <div v-if="show">Hello</div>
7   `
8 }
9
10 test('example', async () => {
11   const wrapper = mount(Component)
12   wrapper.find('button').trigger('click')
13
14   // fails... DOM isn't updated!
15   expect(wrapper.html()).toContain('Hello')
16 })
17
```

① 0 ▲ 0 1 change; before #34 58 seconds ago Spaces: 2 UTF-8 LF JavaScript ⌂ ⌂

Vue renders async; runners are not

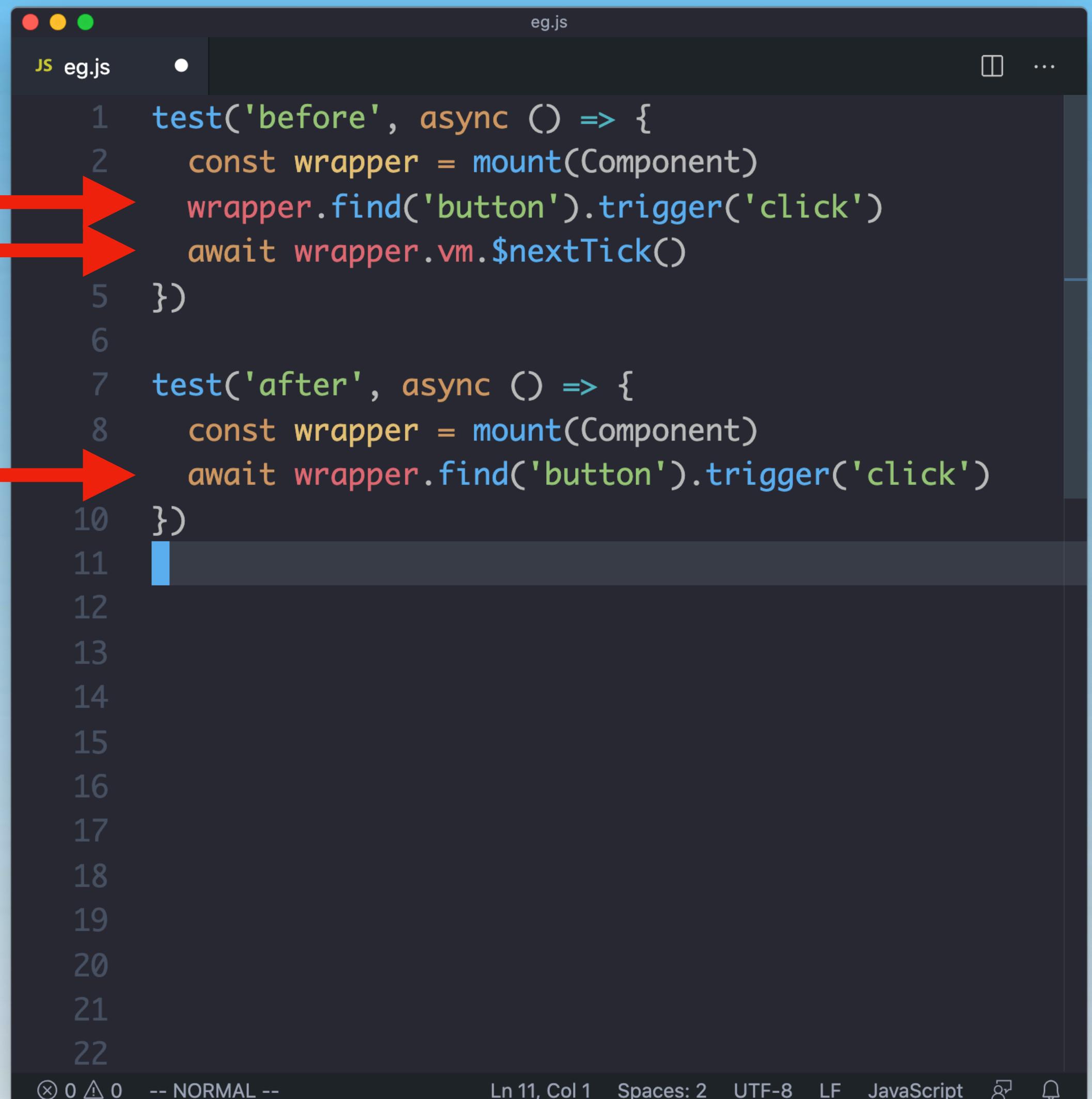
- Solution is to wait for the nextTick
 - Un-necessary overhead

```
eg.js — vue-test-utils
JS eg.js

1 import { mount } from '@vue/test-utils'
2
3 const Component = {
4   template: `
5     <button @click="show = !show" />
6     <div v-if="show">Hello</div>
7   `
8 }
9
10 test('example', async () => {
11   const wrapper = mount(Component)
12   wrapper.find('button').trigger('click')
13
14   // wait for Vue to re-render the DOM
15   await wrapper.vm.$nextTick()
16
17   // fails... DOM isn't updated!
18   expect(wrapper.html()).toContain('Hello')
19 })
20
```

nextTick shorthand

- You can now `await` anything that mutates the DOM
- trigger
- setProps
- setData



The screenshot shows a code editor window titled "eg.js" containing two test cases. Red arrows point from the bulleted list on the left to the corresponding code snippets in the editor.

```
JS eg.js
1  test('before', async () => {
2    const wrapper = mount(Component)
3    wrapper.find('button').trigger('click')
4    await wrapper.vm.$nextTick()
5  })
6
7  test('after', async () => {
8    const wrapper = mount(Component)
9    await wrapper.find('button').trigger('click')
10   })
11
12
13
14
15
16
17
18
19
20
21
22
```

Ln 11, Col 1 Spaces: 2 UTF-8 LF JavaScript

No more createLocalVue (1)

- Vue 2: plugins are installed by *mutating* the global Vue instance
- Not ideal; when we unit test, we want to *isolate*.

```
store.js
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3 Vue.use(Vuex)
4
5 export const createStore = () => {
6   return new Vuex.Store({
7     // ...
8   })
9 }
10

eg.js
1 import { createStore } from './store.js'
2
3 test('does something with Vuex', async () => {
4   const wrapper = mount(Component, {
5     store: createStore()
6   })
7
8   await wrapper.find('button').trigger('click')
9
10  expect(wrapper.html()).toContain('Count: 1')
11 }
12 )
```

store.js

eg.js

Vue

Vuex

use

createStore

new Vuex.Store

// ...

Component

button

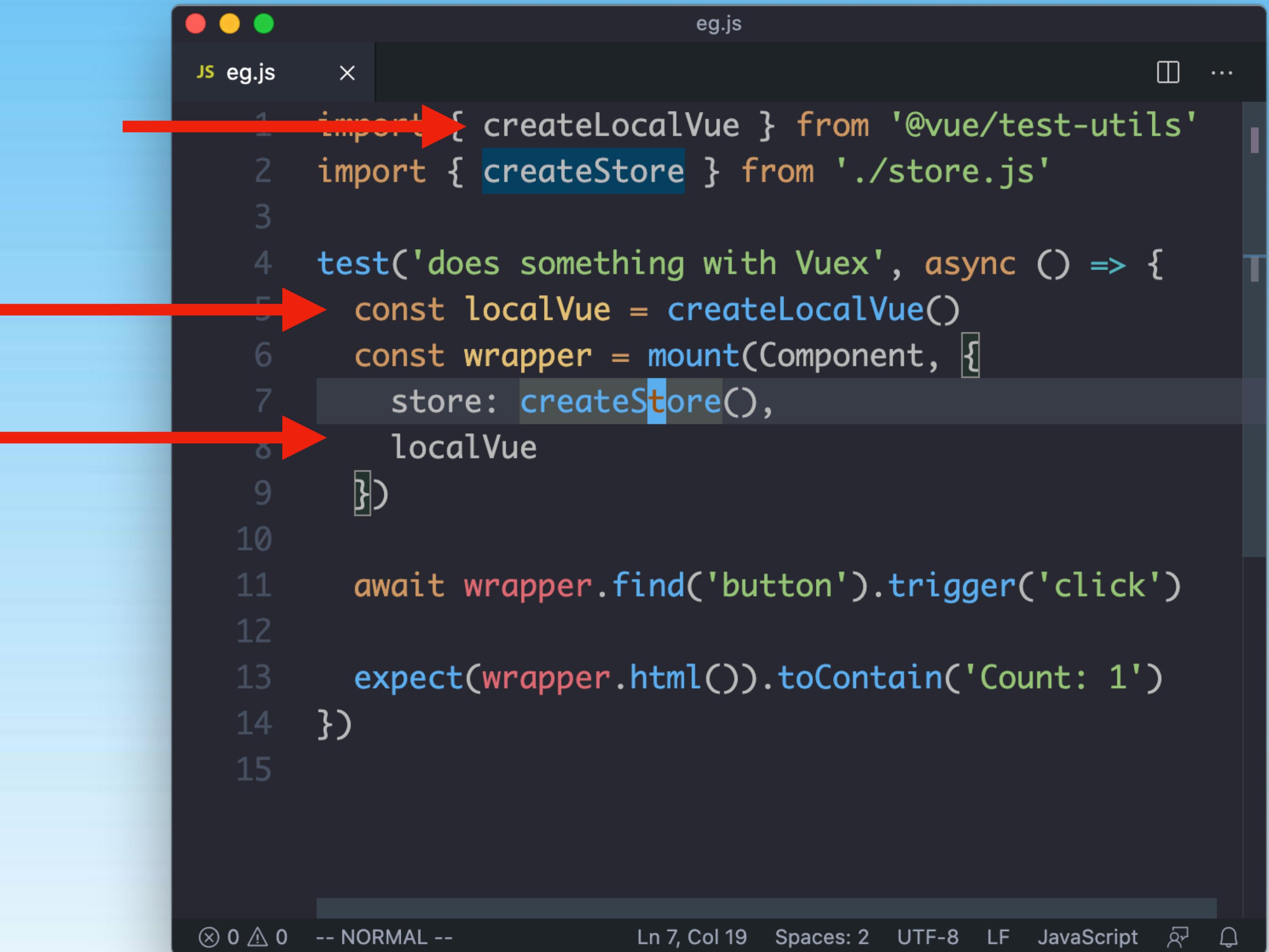
html

Count: 1

Ln 10, Col 1 Spaces: 2 UTF-8 LF JavaScript

No more createLocalVue (2)

- localVue was introduced to solve this problem
- worked pretty well



```
JS eg.js
1 import { createLocalVue } from '@vue/test-utils'
2 import { createStore } from './store.js'
3
4 test('does something with Vuex', async () => {
5   const localVue = createLocalVue()
6   const wrapper = mount(Component, {
7     store: createStore(),
8     localVue
9   })
10
11   await wrapper.find('button').trigger('click')
12
13   expect(wrapper.html()).toContain('Count: 1')
14 })
15
```

No more `createLocalVue` (3)

- Each Vue app and its plugins, directives etc are now *local by default*

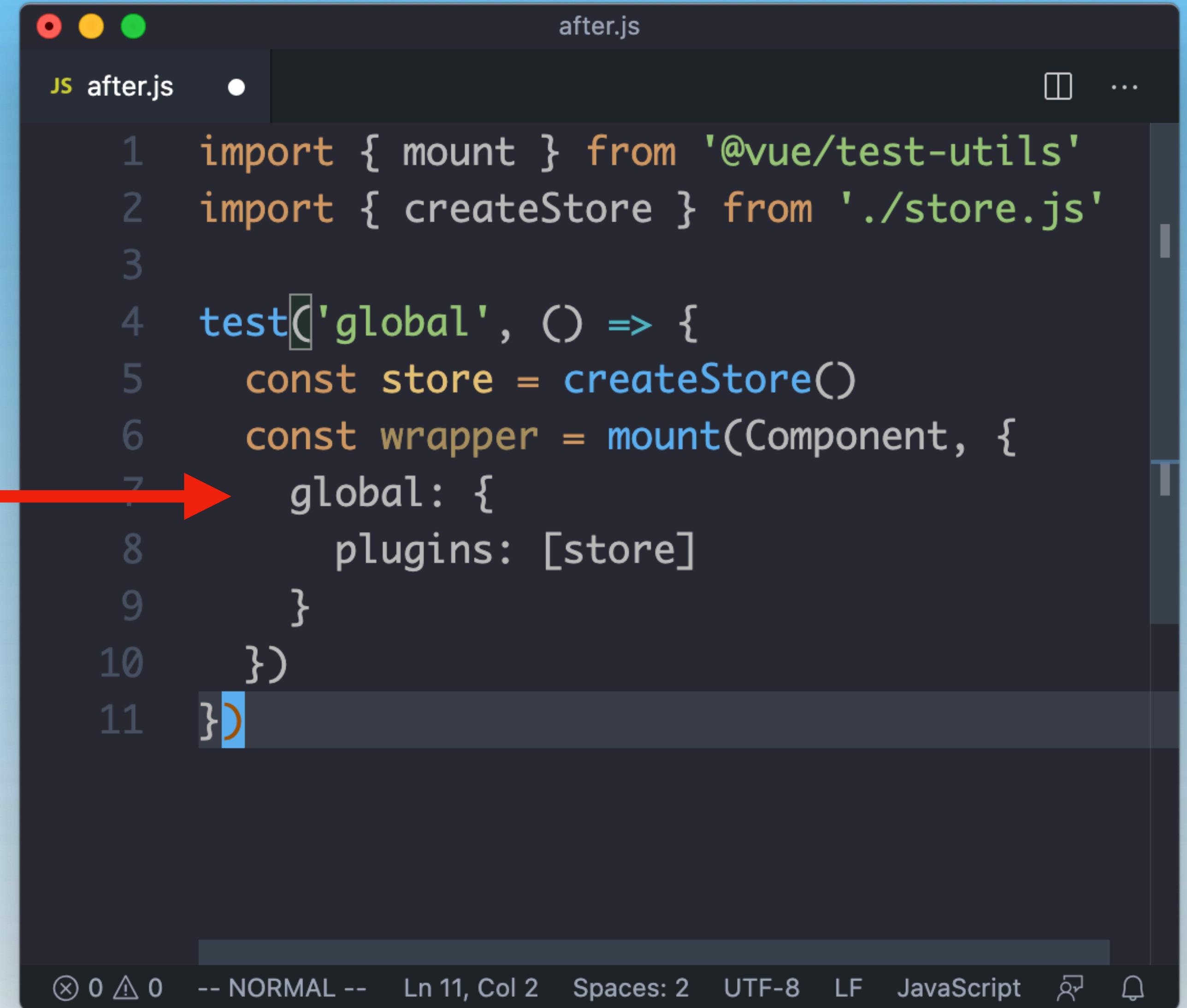
```
store.js
1 import Vue from 'vue'
2 import Vuex from 'vuex'
3 import App from './App.vue'
4 import { createStore } from './store'
5
6 Vue.use(Vuex)
7 const store = createStore()
8
9 new Vue({
10   store,
11   render: h => h(App)
12 }).mount('#app')

after.js
1 import { createApp } from 'vue'
2 import { createStore } from './store'
3 import App from './App.vue'
4
5 const store = createStore()
6
7 // has Vuex installed
8 const app1 = createApp(App)
9 app.use(store)
10 app.mount('#app')
11
12 // does not have Vuex installed
13 // isolated!
14 const app2 = createApp(App)
15
```

The image shows a code editor with two tabs: "store.js" and "after.js". The "store.js" tab contains code for setting up a Vue application with Vuex. The "after.js" tab shows how the code would look if Vuex were not installed, demonstrating the "isolated" nature of the setup.

No more createLocalVue (4)

- Now you install plugins using the global mounting option.



```
JS after.js
1 import { mount } from '@vue/test-utils'
2 import { createStore } from './store.js'
3
4 test('global', () => {
5   const store = createStore()
6   const wrapper = mount(Component, {
7     global: {
8       plugins: [store]
9     }
10  })
11})
```

The screenshot shows a terminal window with a dark theme. The file is named 'after.js'. The code demonstrates a Jest test configuration. A red arrow points to line 7, which defines a 'global' object with a 'plugins' key containing the value 'store'. This indicates that the 'store' plugin will be available across all components mounted in this test.

No more createLocalVue (5)

- This is where anything installed on app goes
- mixins
- provide
- components
- directives
- plugins

```
JS after.js
1 import { mount } from '@vue/test-utils'
2 import { createStore } from './store.js'
3
4 test('global', () => {
5   const store = createStore()
6   const wrapper = mount(Component, {
7     global: {
8       plugins: [store],
9       mixins: /* some mixins */,
10      provide: {
11        theme: 'dark'
12      },
13      components: {
14        Foo
15      },
16      directives: {
17        clickAway: {}
18      }
19    }
20  })
21})
22
```

JS after.js

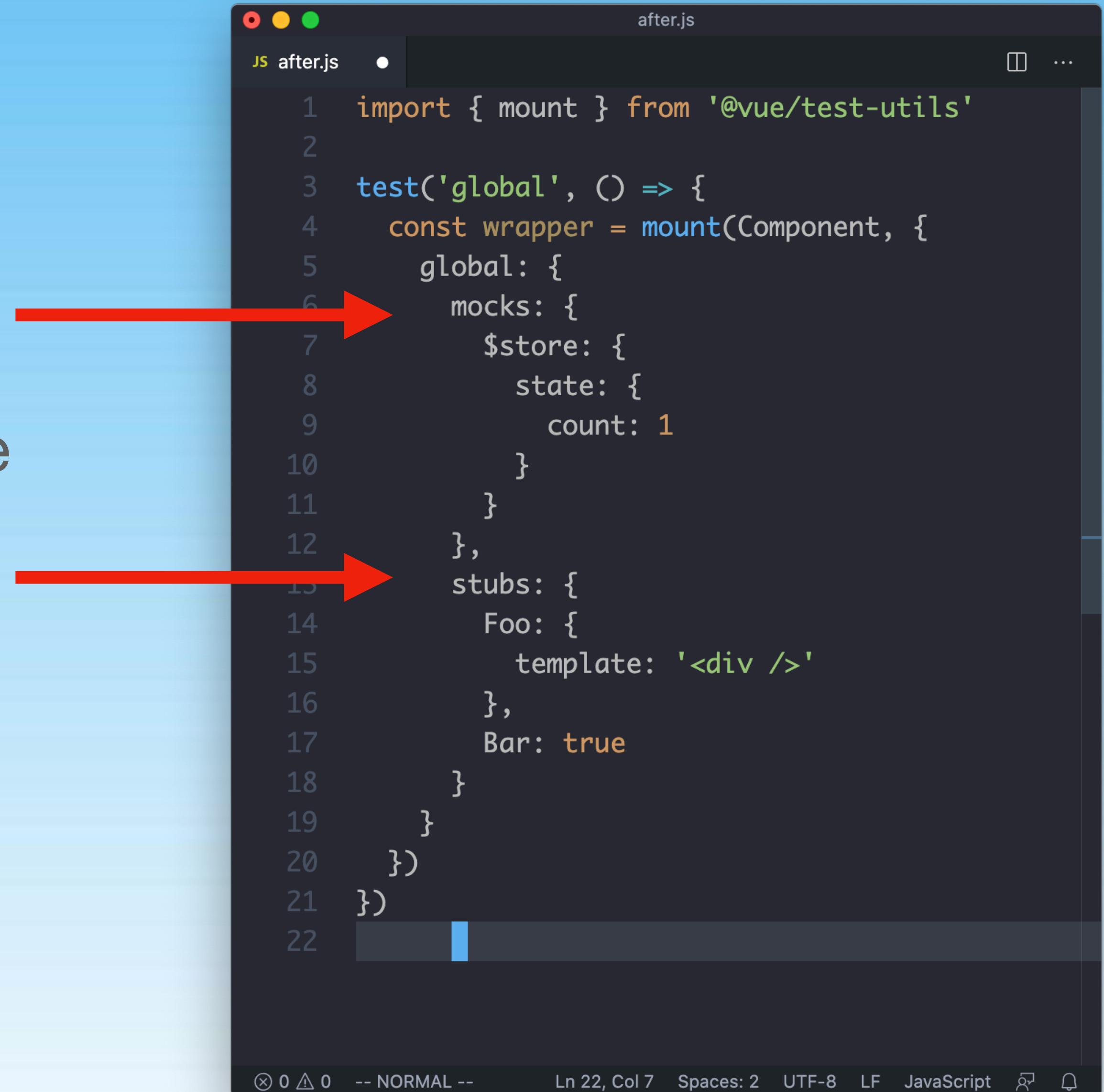
① ② ③ ...

1 import { mount } from '@vue/test-utils'
2 import { createStore } from './store.js'
3
4 test('global', () => {
5 const store = createStore()
6 const wrapper = mount(Component, {
7 global: {
8 plugins: [store],
9 mixins: /* some mixins */,
10 provide: {
11 theme: 'dark'
12 },
13 components: {
14 Foo
15 },
16 directives: {
17 clickAway: {}
18 }
19 }
20 })
21})
22

Ln 3, Col 1 Spaces: 2 UTF-8 LF JavaScript ⚙️ 🔔

No more `createLocalVue` (6)

- This also now includes *stubs* and *mocks*, since those affect all the components, not just the one you are testing, but all of it's children, too.



The screenshot shows a code editor window with a dark theme. The file is named 'after.js'. The code imports 'mount' from '@vue/test-utils' and defines a test function for 'global'. Inside the test function, a component is mounted with a global configuration object. This object includes 'global' (with 'store' and 'state' properties), 'stubs' (with 'Foo' and 'Bar' entries), and 'mocks' (with a 'store' entry). A red arrow points from the explanatory text above to the 'global' object, and another red arrow points from the explanatory text below to the 'stubs' object.

```
JS after.js
1 import { mount } from '@vue/test-utils'
2
3 test('global', () => {
4   const wrapper = mount(Component, {
5     global: {
6       mocks: {
7         $store: {
8           state: {
9             count: 1
10           }
11         }
12       },
13       stubs: {
14         Foo: {
15           template: '<div />'
16         },
17         Bar: true
18       }
19     }
20   })
21 })
22
```

Ln 22, Col 7 Spaces: 2 UTF-8 LF JavaScript ⚡ 🔔

Migration Guide

- Some other small, trivial changes
- See "migration guide" in the VTU next docs

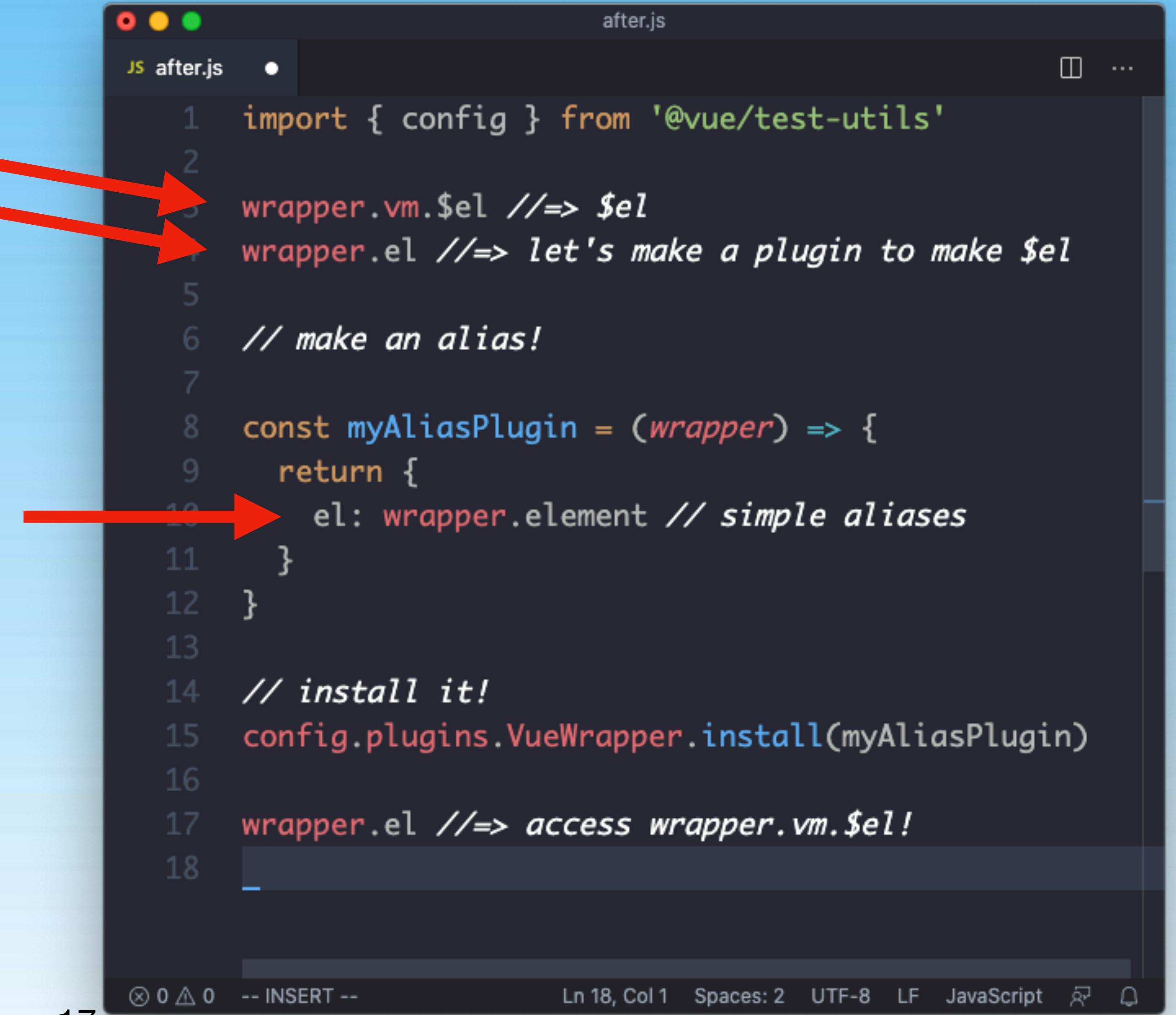
The screenshot shows a browser window displaying the 'Migration' section of the Vue Test Utils documentation. The URL is <https://vue-test-utils.vuejs.org/v2/guide/migration.html#changes>. The page title is 'Vue Test Utils (2.0.0-beta.0)'. The left sidebar includes links for 'Testing with Vue Router', 'Third-party Integration', 'Stubs and Shallow Mount', 'Extending Vue Test Utils' (with 'Plugins' and 'Community and Learning' sub-links), and 'Migration to Vue Test Utils 2' (with 'Migration' and 'Changes' sub-links). The 'Changes' section highlights the transition from 'propsData' to 'props'. It explains that in VTU v1, props were passed via 'propsData' due to internal declaration conflicts, but now they are passed directly via 'props'. A code snippet illustrates the change:

```
const App = {
  props: ['foo']
}

const mount(App, {
  propsData: {
    foo: 'bar'
  }
})
```

New Feature: Plugins!

- Extending Vue Test Utils by writing a plugin
- Function that receives wrapper as the first arg
- Returns methods made available on wrapper

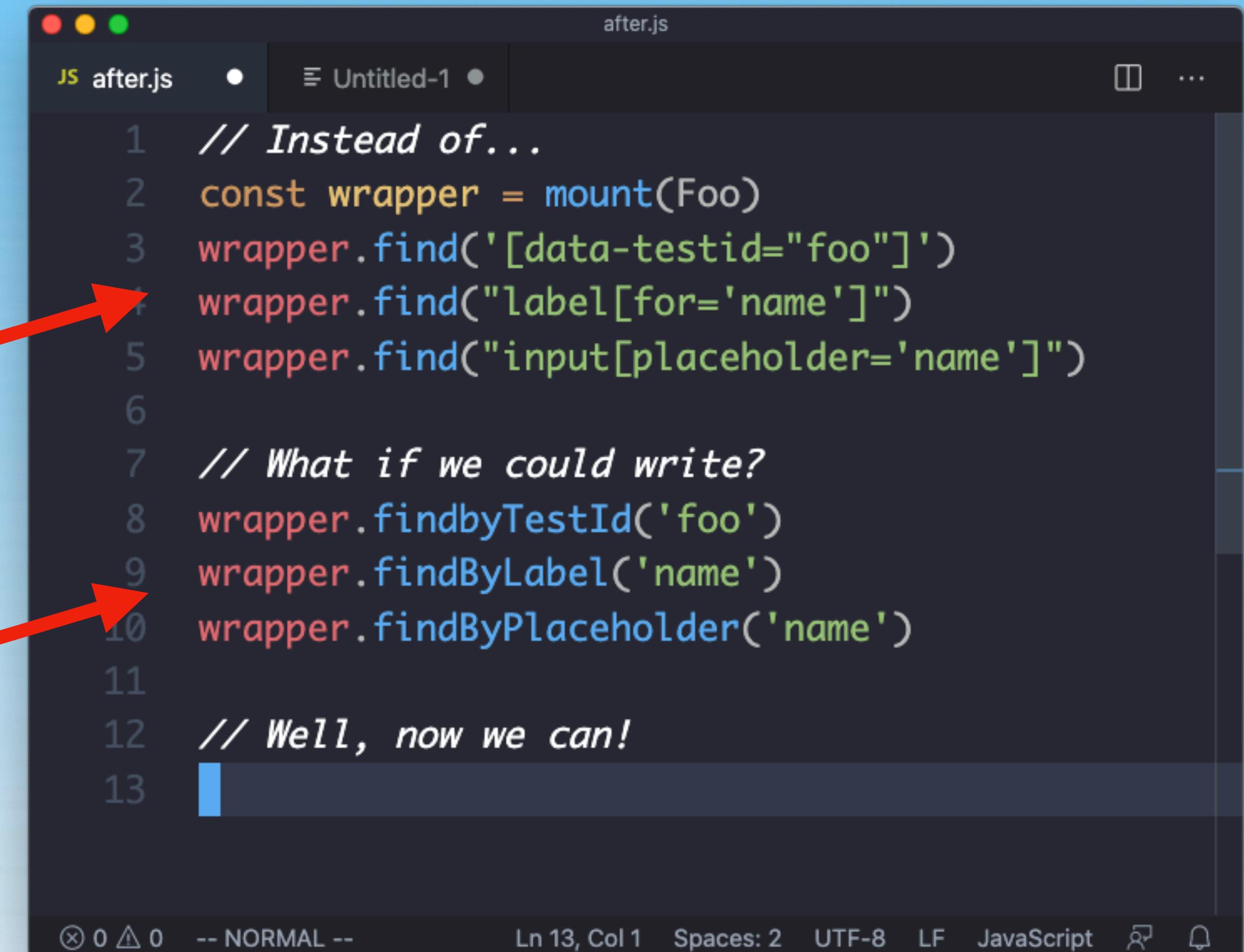


The image shows a code editor window titled 'after.js'. The code is written in JavaScript and demonstrates how to create a plugin for Vue Test Utils. It starts by importing 'config' from '@vue/test-utils'. The code then defines a plugin function 'myAliasPlugin' that takes a 'wrapper' as its argument. Inside the function, it returns an object containing an 'el' key, which is set to 'wrapper.element'. A comment indicates this is a 'simple alias'. The plugin is then installed using 'config.plugins.VueWrapper.install(myAliasPlugin)'. Finally, the code shows that 'wrapper.el' can be used to access 'wrapper.vm.\$el'. Red arrows point from the bullet points in the list above to the corresponding parts of the code: one arrow points to the 'let's make a plugin' comment, and another points to the 'el: wrapper.element' line.

```
1 import { config } from '@vue/test-utils'
2
3 wrapper.vm.$el //=> $el
4 wrapper.el //=> let's make a plugin to make $el
5
6 // make an alias!
7
8 const myAliasPlugin = (wrapper) => {
9   return {
10     el: wrapper.element // simple aliases
11   }
12 }
13
14 // install it!
15 config.plugins.VueWrapper.install(myAliasPlugin)
16
17 wrapper.el //=> access wrapper.vm.$el!
18
```

New Feature: Plugins!

- Can also add new methods to make your tests more expressive.



The screenshot shows a code editor window titled "after.js". The code is a snippet demonstrating the use of a plugin to add new methods to the wrapper object:

```
// Instead of...
const wrapper = mount(Foo)
wrapper.find('[data-testid="foo"]')
wrapper.find("label[for='name']")
wrapper.find("input[placeholder='name']")

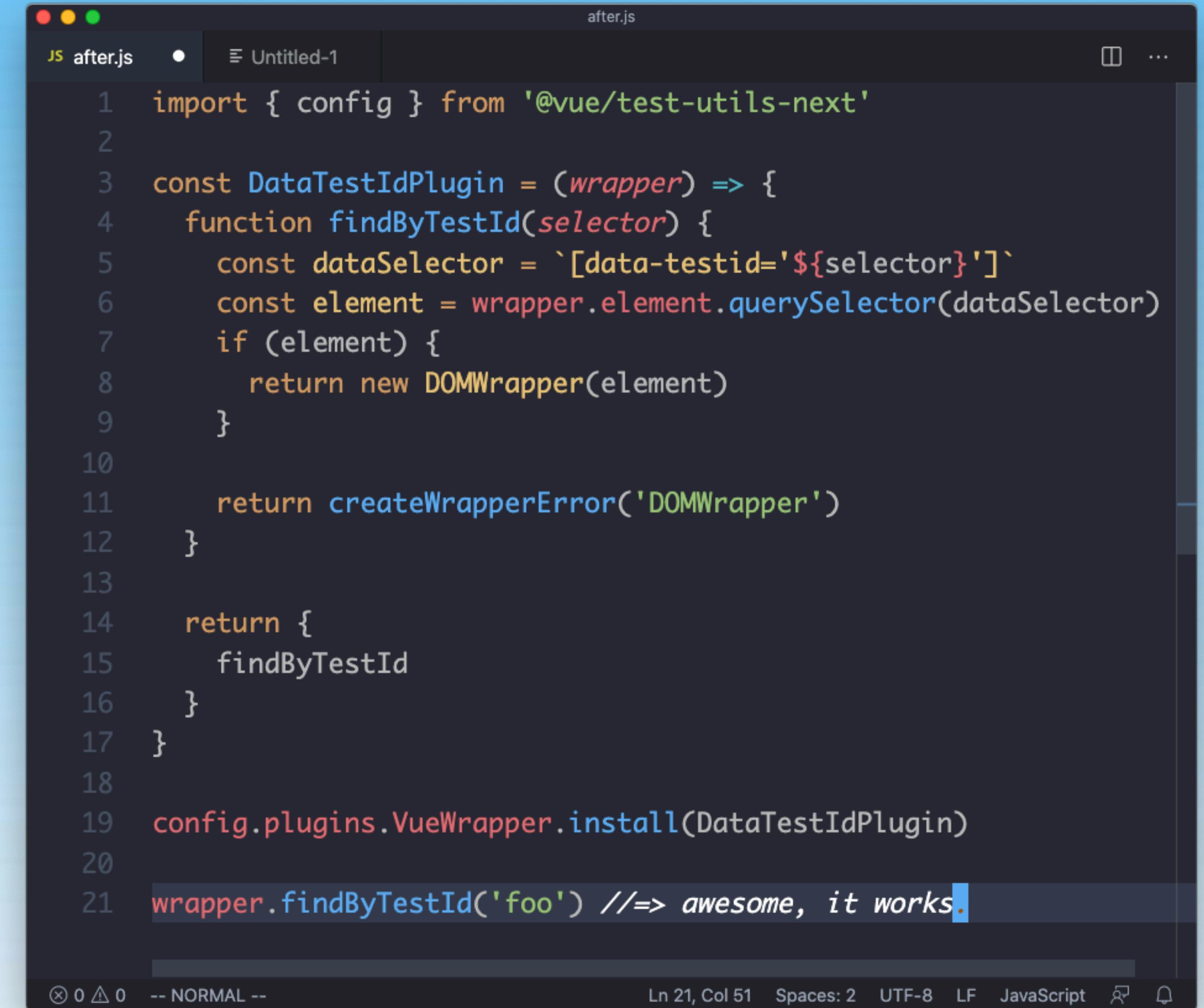
// What if we could write?
wrapper.findByTestId('foo')
wrapper.findByLabel('name')
wrapper.findByPlaceholder('name')

// Well, now we can!
```

Two red arrows point from the text "make your tests more expressive." in the list above to the code snippets in the editor window, specifically pointing at the first two sections of code.

Example: findByTestId

- VueWrapper, DOMWrapper and other utilities are exposed, so you can be very creative.



```
JS after.js • Untitled-1 after.js
1 import { config } from '@vue/test-utils-next'
2
3 const DataTestIdPlugin = (wrapper) => {
4   function findByTestId(selector) {
5     const dataSelector = `[data-testid='${selector}]`
6     const element = wrapper.element.querySelector(dataSelector)
7     if (element) {
8       return new DOMWrapper(element)
9     }
10
11   return createWrapperError('DOMWrapper')
12 }
13
14 return {
15   findByTestId
16 }
17 }
18
19 config.plugins.VueWrapper.install(DataTestIdPlugin)
20
21 wrapper.findByTestId('foo') //=> awesome, it works!
```

Updated Docs

- Greatly simplified.
- More focus on *how*, *why* and *what* to test.

The screenshot shows a browser window displaying the Vue Test Utils documentation at <https://vue-test-utils.vuejs.org/v2/guide/http-requests.html>. The page title is "Vue Test Utils (2.0.0-beta.0)". On the left, there's a sidebar with sections like "Essentials", "Vue Test Utils in depth", "Making HTTP requests", "Extending Vue Test Utils", and "Migration to Vue Test Utils 2". The main content area is titled "Making HTTP requests" and discusses modern test runners' features for testing HTTP requests. It includes a code snippet for a "PostList" component:

```
<template>
<div>
  <button @click="getPosts">Get posts</button>

  <ul>
    <li v-for="post in posts" :key="post.id" data-test="post">
      {{ post.title }}
    </li>
  </ul>
</div>
</template>

<script>
import axios from 'axios'

export default {
  data() {
    return {
      posts: null
    }
  },
  methods: {
```

A New Philosophy

- Stable API.
 - Use RFC process for changes.
- Vue Test Utils are *utilities for testing*.
- Unopinionated, so you can write opinionated tests.
- Support for extending and building on.

Testing Library

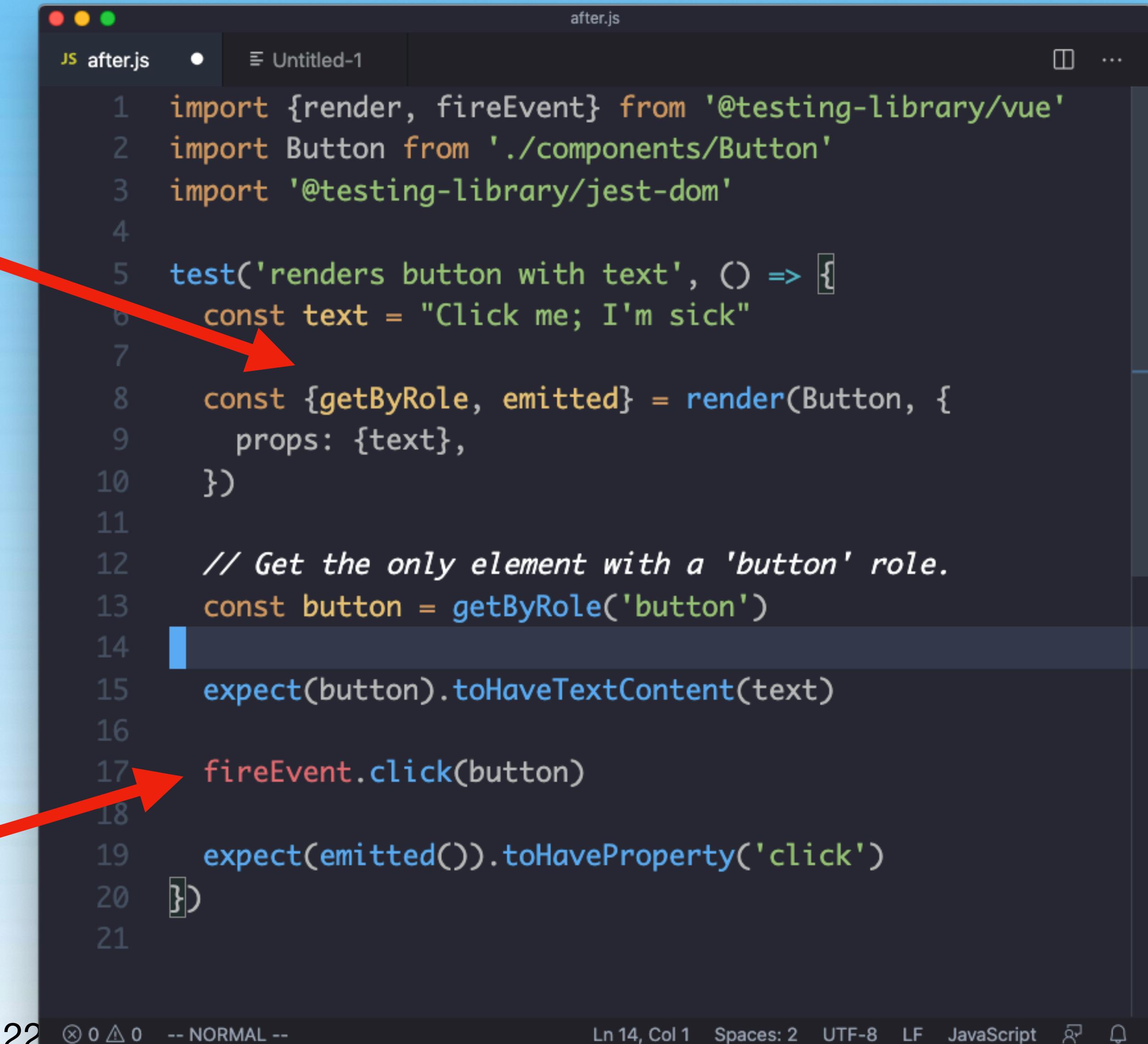
Simple and complete testing utilities that encourage good testing practices



cypress

Case Study: Testing Library

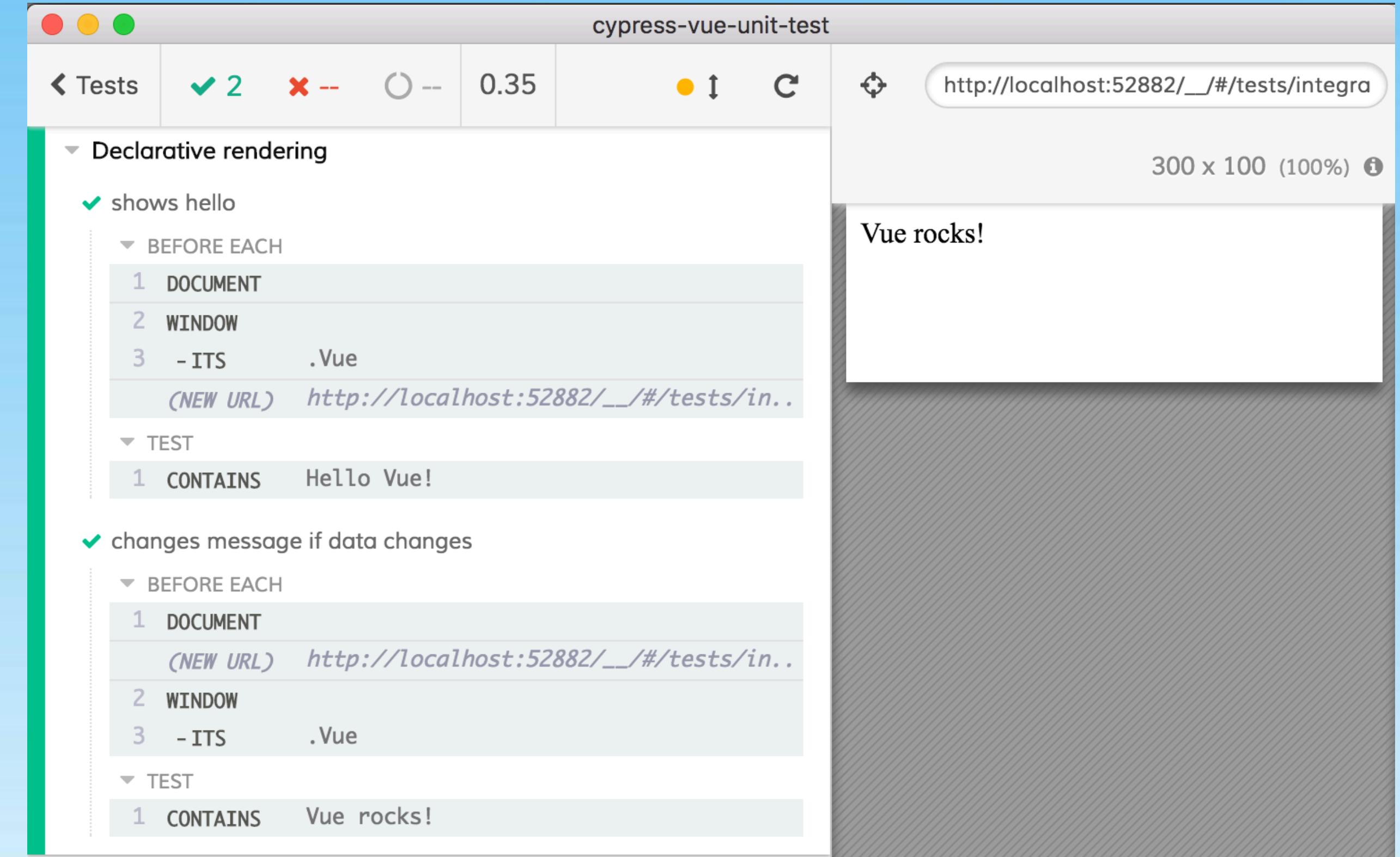
- Write maintainable tests
- Develop with confidence
- Accessible by default
- Opinionated
- Vue Integration by Adria Fontcuberta ([@afontq](#))
- <https://github.com/testing-library/vue-testing-library>



```
JS after.js  • Untitled-1
1 import {render, fireEvent} from '@testing-library/vue'
2 import Button from './components/Button'
3 import '@testing-library/jest-dom'
4
5 test('renders button with text', () => {
6   const text = "Click me; I'm sick"
7
8   const {getByRole, emitted} = render(Button, {
9     props: {text},
10   })
11
12   // Get the only element with a 'button' role.
13   const button = getByRole('button')
14
15   expect(button).toHaveTextContent(text)
16
17   fireEvent.click(button)
18
19   expect(emitted()).toHaveProperty('click')
20 })
21
```

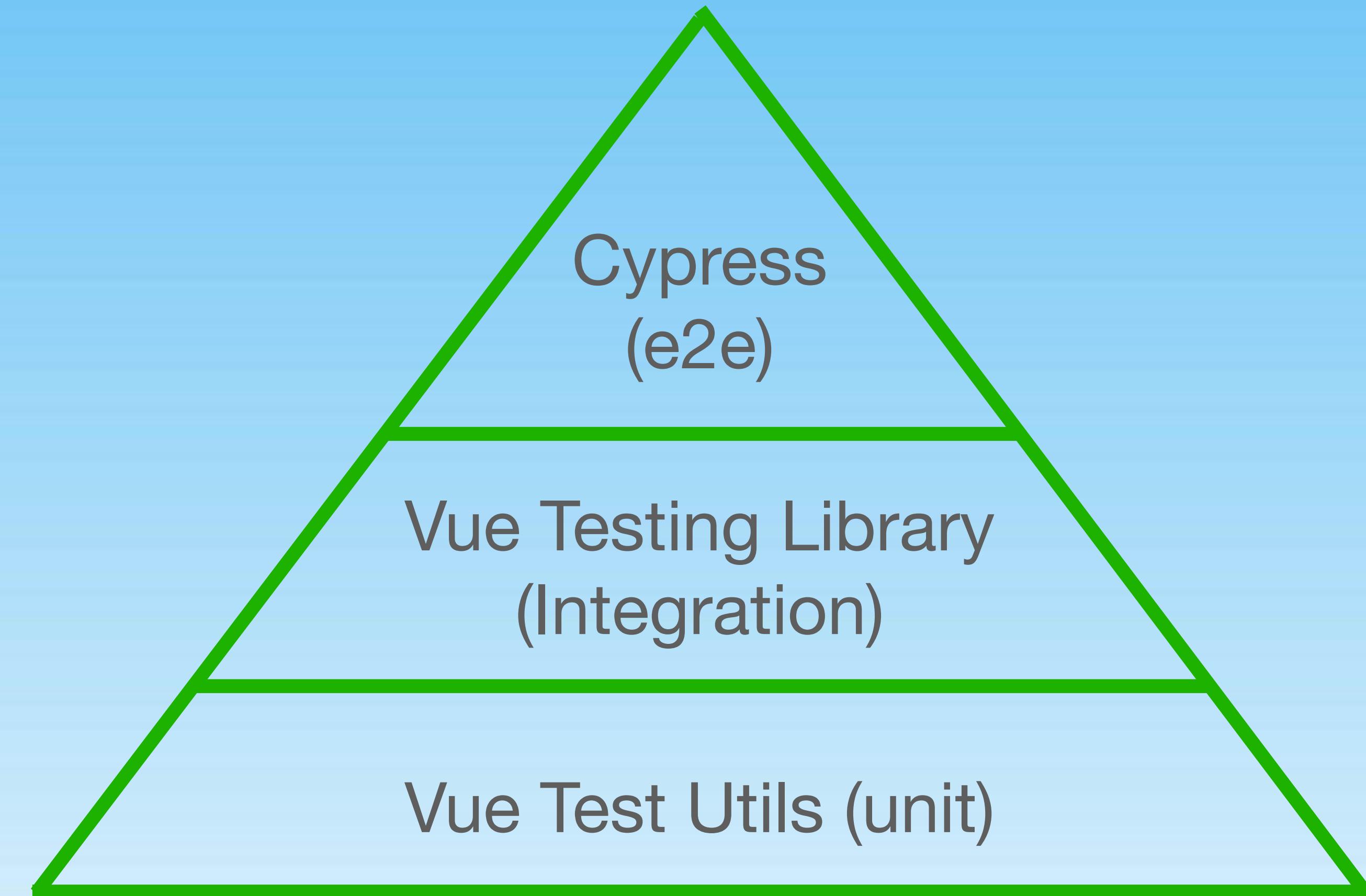
Case Study: Cypress

- Runs in a real browser
- Opinionated. *Tests like your users do.*
- Powered by Test Utils.
- Early days, but promising!
Cypress is awesome.
- Vue Integration by Jess Sachs (@_jessicasachs)
- <https://www.youtube.com/watch?v=a-9axZxTFZU>



Which tool should I use?

- Vue Testing Pyramid
- You can basically write the same tests with any of these tools
- Personal preference
- Simple utilities library vs opinionated testing platform



Where to next?

- Official documentation
- Some simple guides on getting started
- API reference

The screenshot shows a web browser displaying the official documentation for Vue Test Utils (v2.0.0-beta.0). The URL in the address bar is <https://vue-test-utils.vuejs.org/v2/guide/introduction.html>. The page title is "Vue Test Utils (2.0.0-beta.0)". The left sidebar contains a navigation menu with sections like "Essentials", "Vue Test Utils in depth", and "Extending Vue Test Utils". The main content area starts with a welcome message: "Welcome to Vue Test Utils, the official testing utility library for Vue.js!". It then provides a brief overview: "This is the documentation for Vue Test Utils v2, which targets Vue 3." Below this, there's a section titled "What is Vue Test Utils?" with a description: "Vue Test Utils (VTU) is a set of utility functions aimed to simplify testing Vue.js components. It provides some methods to mount and interact with Vue components in an isolated manner." A code example is shown in a dark box:

```
import { mount } from '@vue/test-utils'

// The component to test
const MessageComponent = {
  template: '<p>{{ msg }}</p>',
  props: ['msg'],
}

test('displays message', () => {
  const wrapper = mount(MessageComponent, {
    props: {
      msg: 'Hello world'
    }
  })

  // Assert the rendered text of the component
  expect(wrapper.text()).toContain('Hello world')
})
```

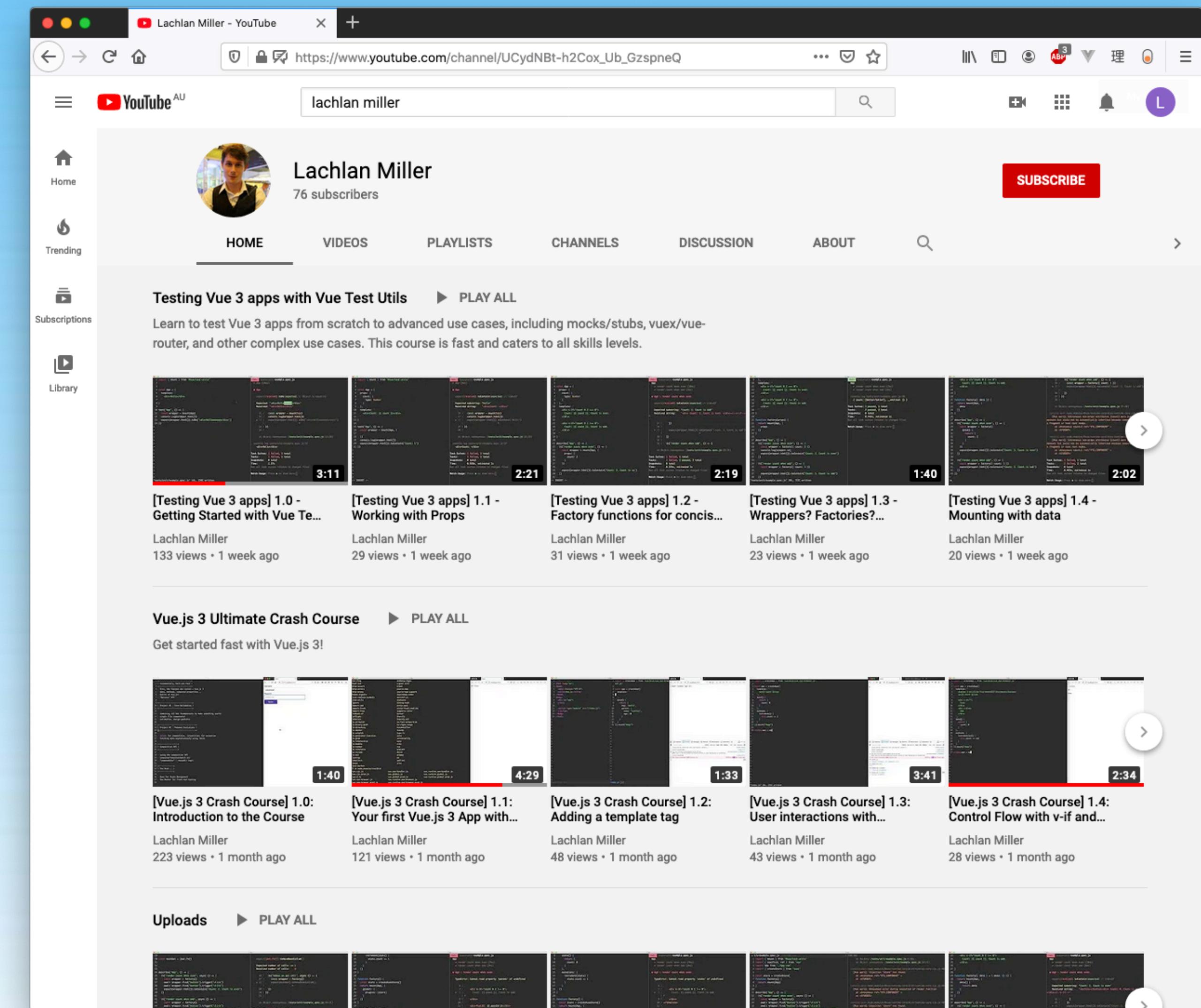
Where to next?

- Vue Testing Handbook
- Updated for Vue 3 + Vue Test Utils v2
- <https://lmiller1990.github.io/vue-testing-handbook/v3/>

The screenshot shows a web browser displaying the Vue Testing Handbook (Vue.js 3) at the URL <https://lmiller1990.github.io/vue-testing-handbook/v3/>. The page title is "Vue Testing Handbook (Vue.js 3)". On the left, there is a sidebar with a blue header containing the text "Vue.js", "The Composition API", and a promotional message for a course. Below the sidebar is a list of navigation links: Welcome, What is this guide?, Further Reading, Setting up for TDD, Rendering Components, Testing Props, Computed Properties, Simulating user input, Testing emitted events, Mocking global objects, Stubbing components, Finding elements and components, Testing Vuex, Vuex - Mutations, Vuex - Actions, Vuex - Getters, Vuex in components - \$state and getters, Vuex in components - mutations and actions, Vue Router, and Composition API. The main content area starts with a section titled "# What is this guide?", followed by a welcome message, a general introduction to the handbook's purpose, and a list of topics covered. It also includes sections on testing various scenarios like Vuex and Jest API usage.

Where to next?

- Video lectures on YouTube
- <https://www.youtube.com/watch?v=0IV4dVYOyBw&list=PLC2LZCNWKL9ahK1IoODqYxKu5aA9T5IOA>



Lachlan Miller | State of Vue Testing
<https://lachlan-miller.me/vue-toronto-2020>

Where to next?

- I made a course about TypeScript, Composition API.
- Write tests for EVERYTHING!
- <https://vuejs-course.com/>
- VUE_TORONTO_25_OFF
- Also on Udemy, if you like Udemy. But you should support independent creators!

Lachlan Miller - YouTube X Lachlan Miller X Online Courses - Anytime, Anyw X +

U Udemy Categories vue composition api Instructor My courses

9,091 results for "vue composition api"

Not sure? All courses have a 30-day money-back guarantee

Filter Most Relevant 9,091 results

Topic

- Vue JS (4)
- Vuex (2)
- JavaScript (1)
- Web Development (1)

Level

- All Levels (4,179)
- Beginner (2,892)
- Intermediate (1,825)
- Expert (194)

Language

Price

Features

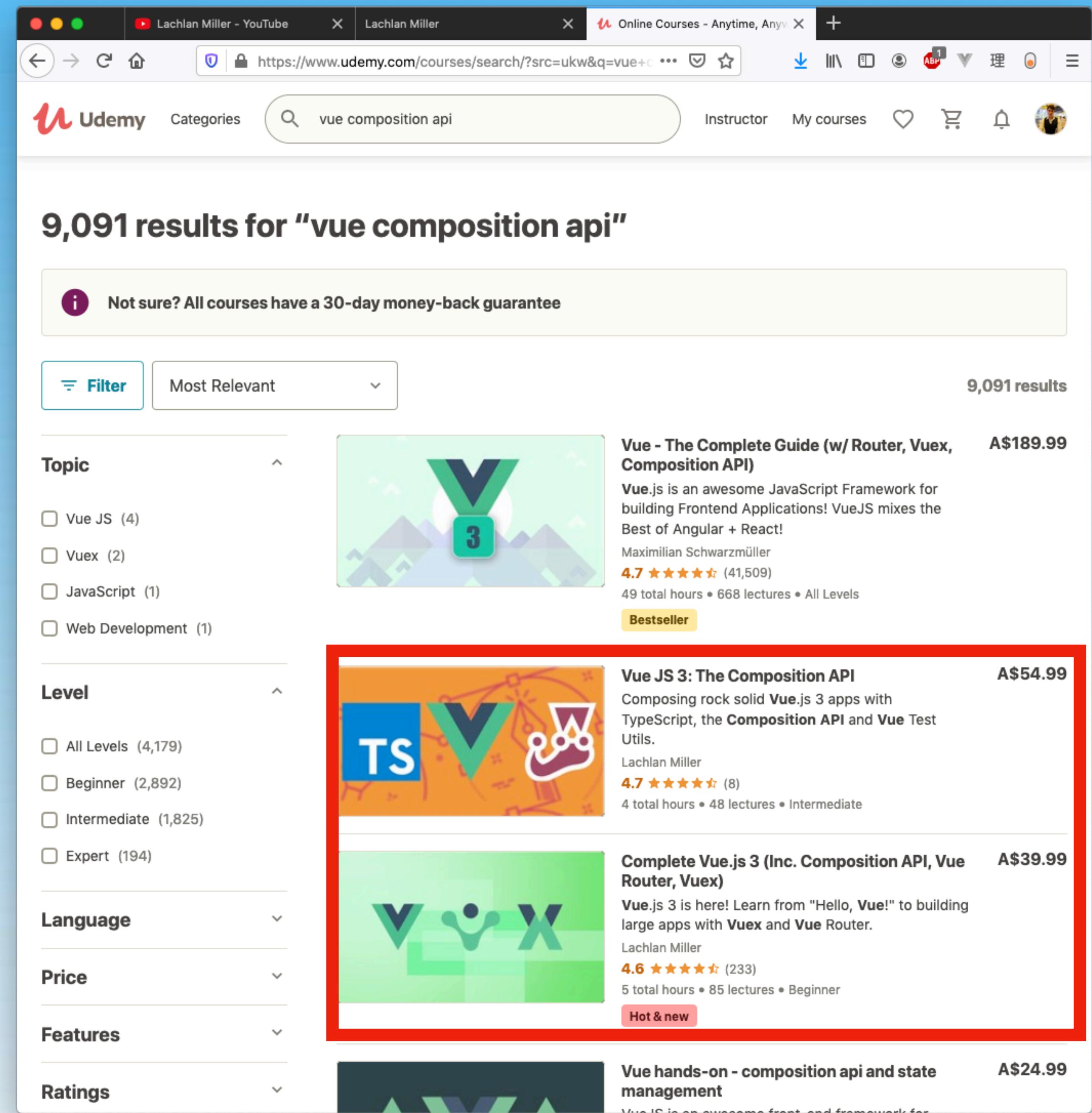
Ratings

Vue - The Complete Guide (w/ Router, Vuex, Composition API) A\$189.99
Vue.js is an awesome JavaScript Framework for building Frontend Applications! VueJS mixes the Best of Angular + React!
Maximilian Schwarzmüller
4.7 ★★★★★ (41,509)
49 total hours • 668 lectures • All Levels
Bestseller

Vue JS 3: The Composition API A\$54.99
Composing rock solid Vue.js 3 apps with TypeScript, the **Composition API** and **Vue Test Utils**.
Lachlan Miller
4.7 ★★★★★ (8)
4 total hours • 48 lectures • Intermediate

Complete Vue.js 3 (Inc. Composition API, Vue Router, Vuex) A\$39.99
Vue.js 3 is here! Learn from "Hello, Vue!" to building large apps with **Vuex** and **Vue Router**.
Lachlan Miller
4.6 ★★★★★ (233)
5 total hours • 85 lectures • Beginner
Hot & new

Vue hands-on - composition api and state management A\$24.99
Vue.js is an awesome front-end framework for



Lachlan Miller | State of Vue Testing
<https://lachlan-miller.me/vue-toronto-2020>

The State of Vue Testing

**Presented by Lachlan Miller
Vue Toronto 2020**