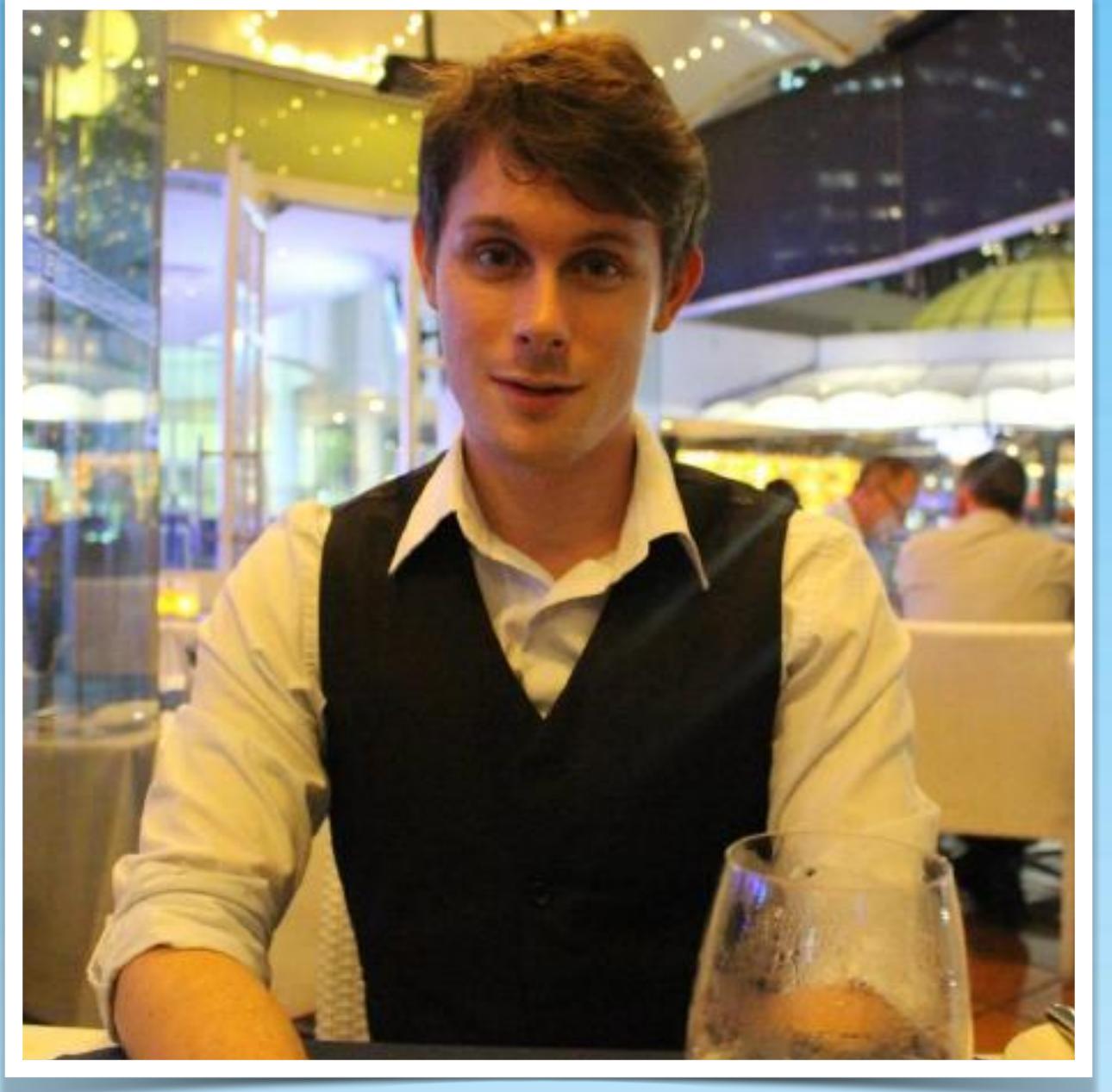


# **The State of Vue Testing**

**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: @Lachlan19900
- GitHub: lmiller1990



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

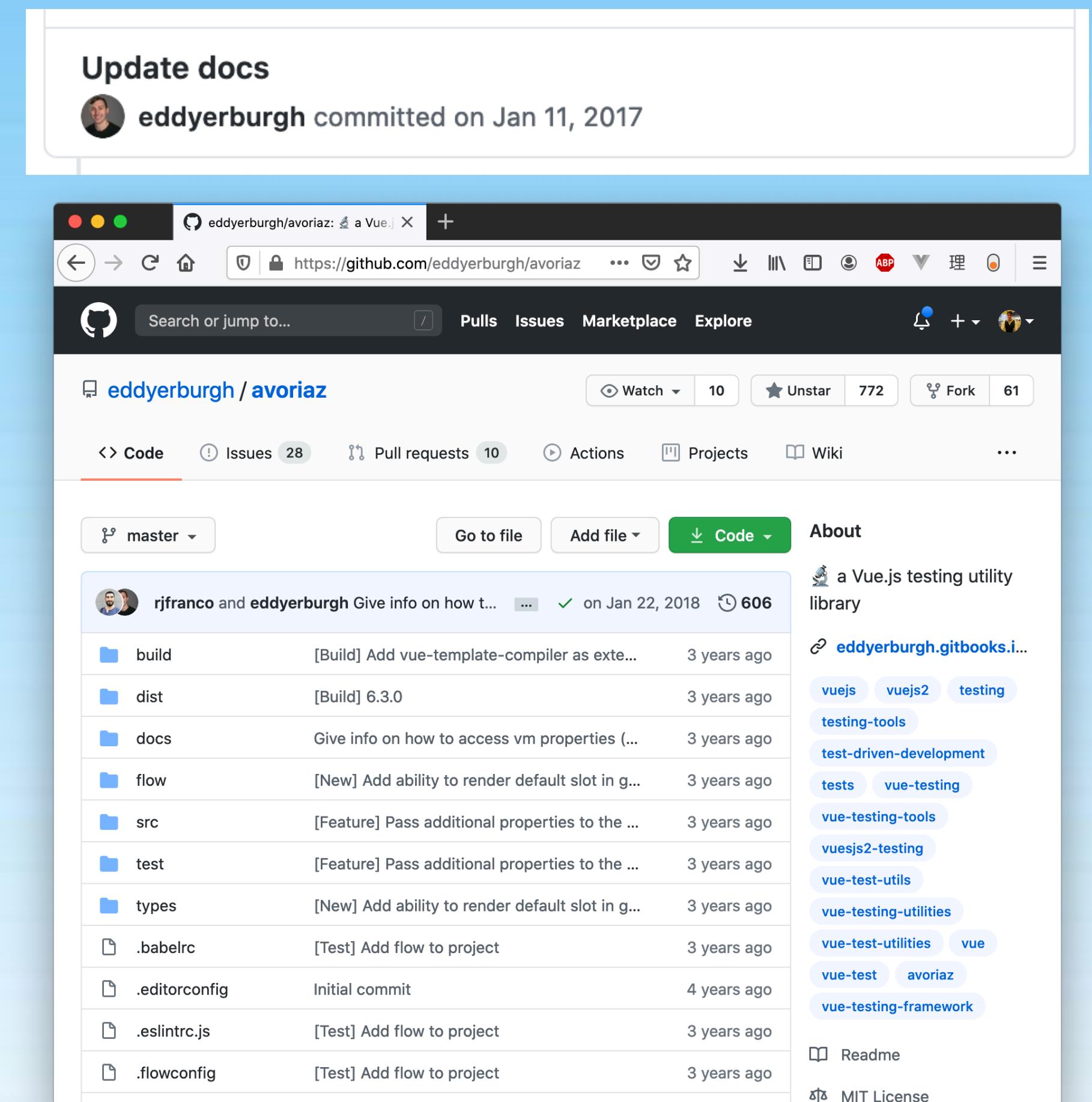
# Agenda

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...

- 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...

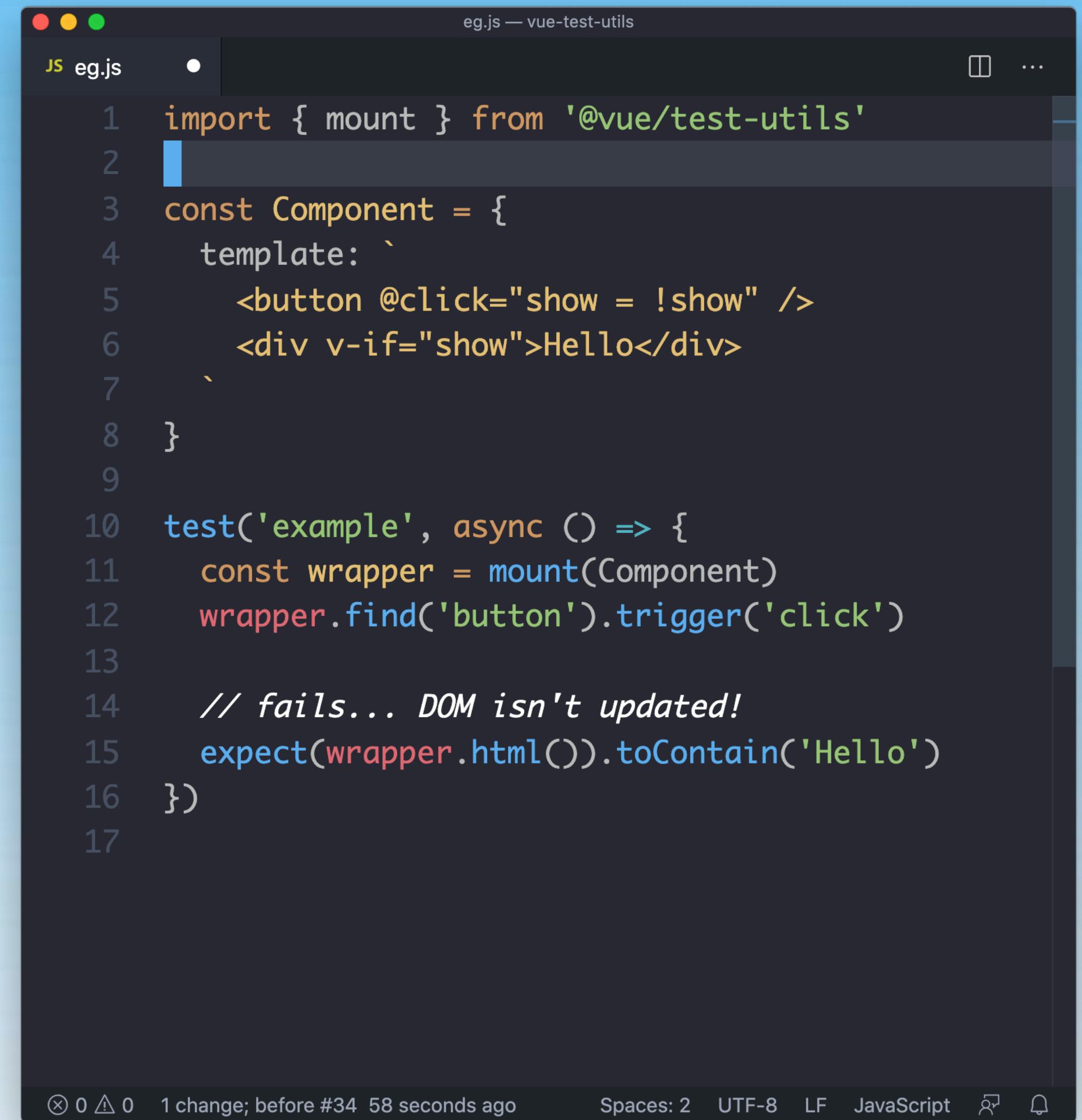
- Avoriaz used as a base for Vue Test Utils
- Naming similar to "React Test Utils"
- *Utilities for testing Vue components*

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

Initial commit 🚀

# Vue renders async; runners are not

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



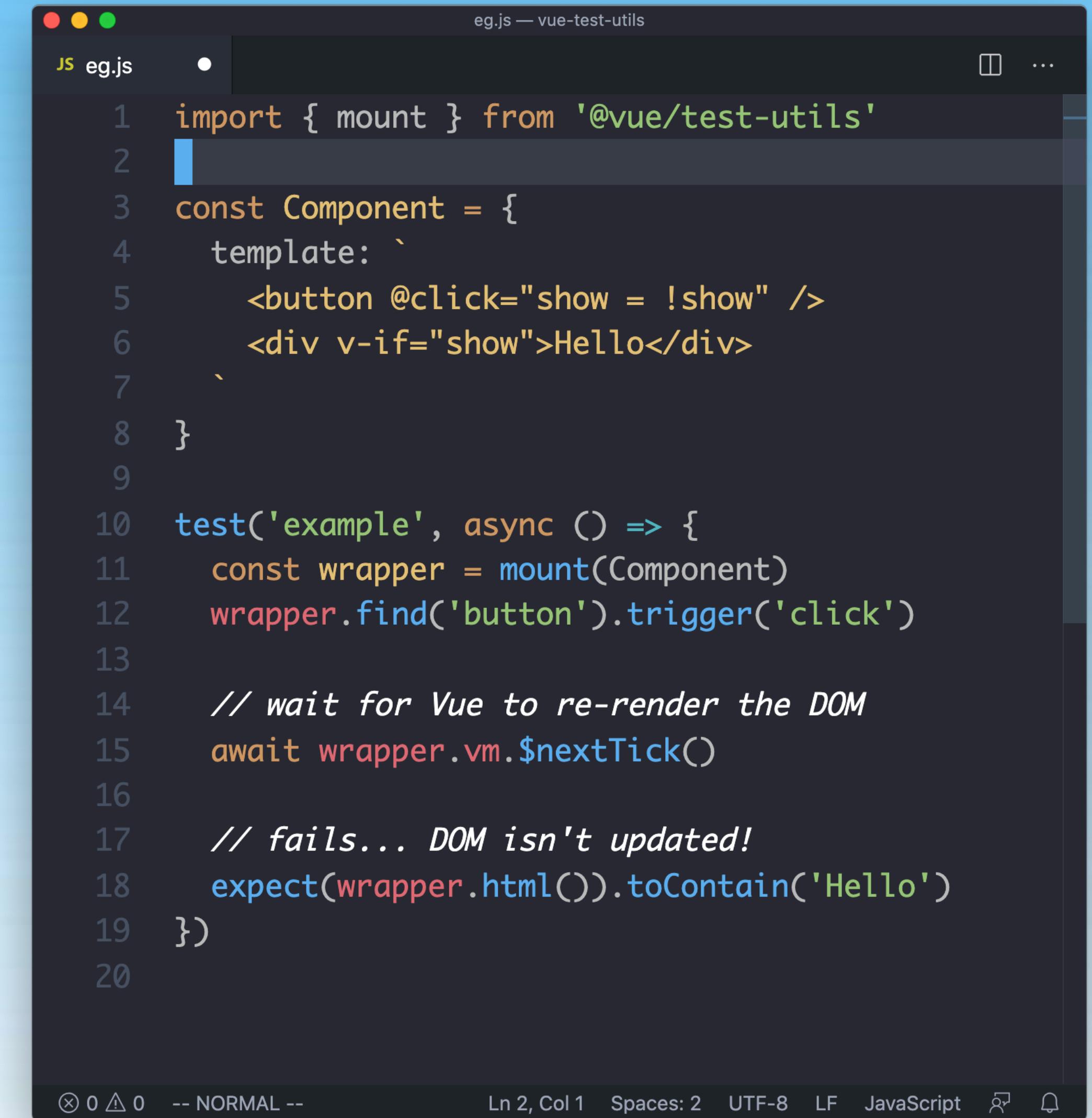
A screenshot of a code editor window titled "eg.js — vue-test-utils". The file contains the following code:

```
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   // fails... DOM isn't updated!
15   expect(wrapper.html()).toContain('Hello')
16 })
17
```

The code defines a component named `Component` with a template that includes a button and a conditional div. A test is written using `@vue/test-utils` to mount the component, trigger a click on the button, and then check if the DOM contains the text "Hello". The test fails because the DOM hasn't been updated yet.

# 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
```

① 0 ▲ 0 -- NORMAL -- Ln 2, Col 1 Spaces: 2 UTF-8 LF JavaScript ⌂ ⌂

# nextTick shorthand

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

```
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 -- NORMAL -- 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*.

The image shows a code editor with two tabs open: 'store.js' and 'eg.js'. The 'store.js' tab contains code for creating a Vuex store:

```
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
```

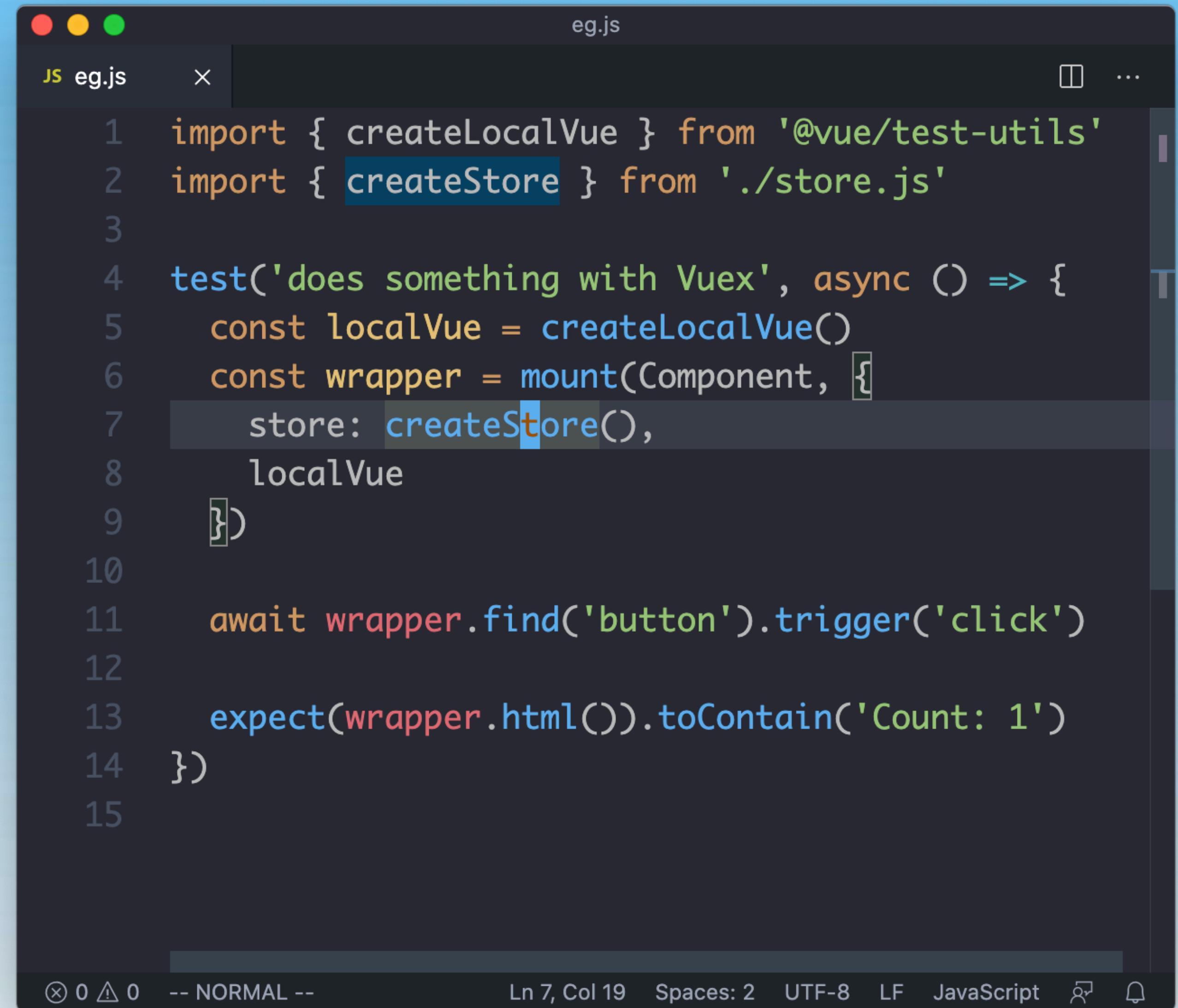
The 'eg.js' tab contains a Jest test for a component using the store:

```
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
```

At the bottom of the editor, status bar text includes: '@ 0 △ 0 Already at oldest change Ln 10, Col 1 Spaces: 2 UTF-8 LF JavaScript ⚡ 🔍 🔞

# No more createLocalVue (2)

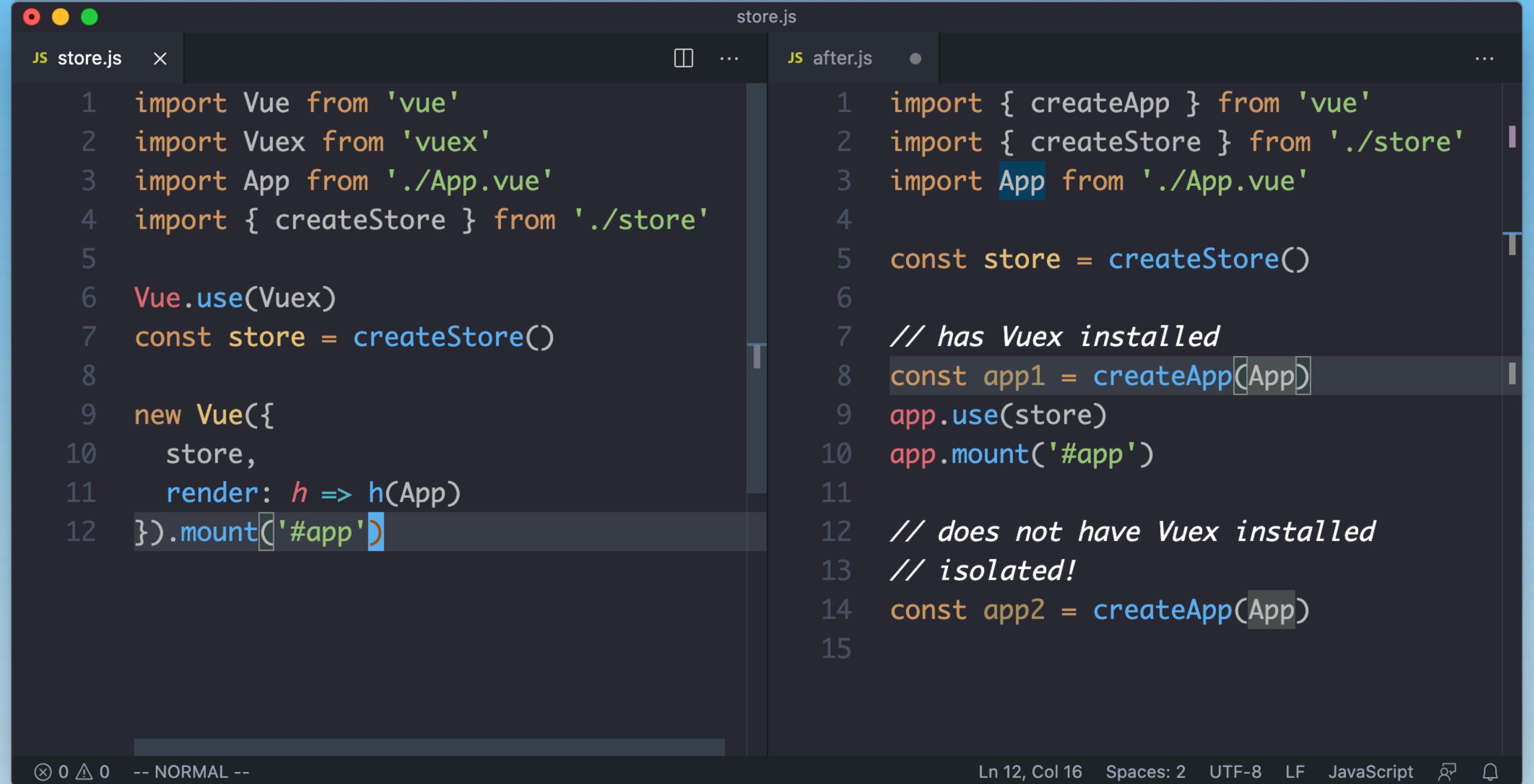
- localVue was introduced to solve this problem
- worked pretty well



```
eg.js
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 app1.use(store)
10 app1.mount('#app')
11
12 // does not have Vuex installed
13 // isolated!
14 const app2 = createApp(App)
15
```

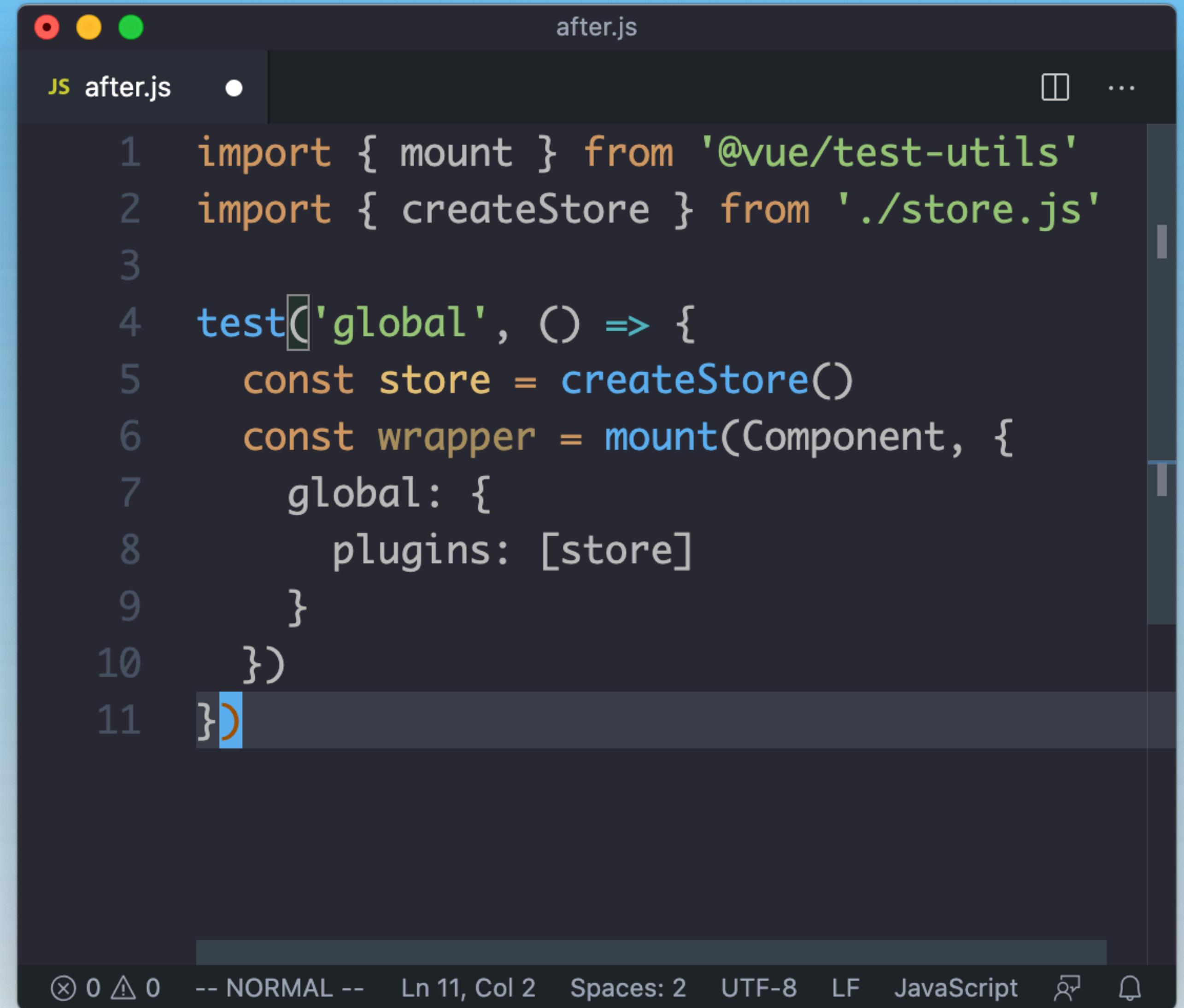
store.js

after.js

Ln 12, Col 16 Spaces: 2 UTF-8 LF JavaScript ⚙️ 🔔

# No more createLocalVue (4)

- Now you install plugins using the global mounting option.



The terminal window shows a file named 'after.js' with the following content:

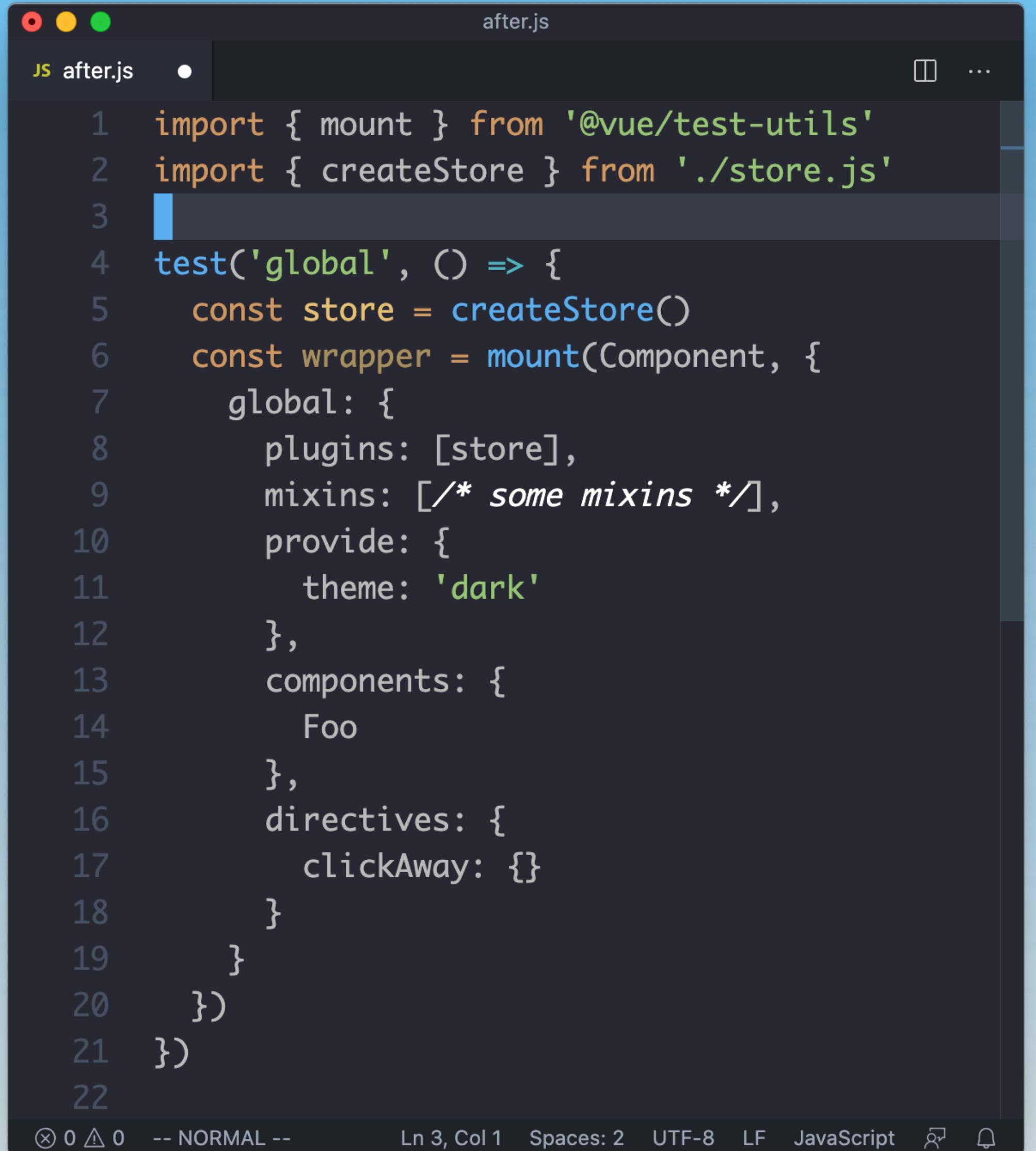
```
JS after.js
1 import { mount } from '@vue/test-utils'
2 import { createStore } from './store.js'
3
4 test('global', () => {
5   const store = createStore()
6   const wrapper = mount(Component, {
7     global: {
8       plugins: [store]
9     }
10  })
11})
```

The status bar at the bottom of the terminal window indicates the following information:

- Line count: 12
- Character count: 0
- Warning count: 0
- Mode: NORMAL
- Line number: Ln 11, Col 2
- Spaces: 2
- Encoding: UTF-8
- Line separator: LF
- File type: JavaScript
- User icon: A small user icon
- Notification icon: A bell icon

# 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
```

The screenshot shows a code editor window titled "after.js". The code is a Jest test configuration object. It imports `mount` from `@vue/test-utils` and `createStore` from `./store.js`. The test itself is named "global" and uses the `createStore` function to create a store. It then mounts a component and provides a "theme" of "dark" to the global context. The component being tested is named "Foo". The code editor interface includes tabs for JS and after.js, a status bar at the bottom, and a dark-themed UI.

# 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 the 'mount' function from '@vue/test-utils' and defines a test for a component named 'global'. The test uses the 'global' wrapper to mount the component with a store that has a state of { count: 1 }. It also uses stubs for 'Foo' and 'Bar' components. The 'Foo' stub has an empty template, and the 'Bar' stub is set to true. The code editor interface includes status bars at the bottom showing file statistics like 'Ln 22, Col 7' and encoding like 'UTF-8'.

```
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
```

# Migration Guide

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

The screenshot shows a web 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 'Migration' section is highlighted in green. Below it, the 'Changes' section is also highlighted in green. A code snippet at the bottom shows the transition from using 'propsData' to using 'props'.

**Migration**

A review of changes VTU 1 -> VTU 2, and some code snippets to showcase required modifications. If you encounter a bug or difference in behavior not documented here, please open an issue.

**Changes**

# `propsData` is now `props`

In VTU v1, you would pass props using the `propsData` mounting option. This was confusing, because you declare props inside of the `props` option in your Vue components. Now you can pass `props` using the `props` mounting option. `propsData` is and will continue to be supported for backwards compatibility.

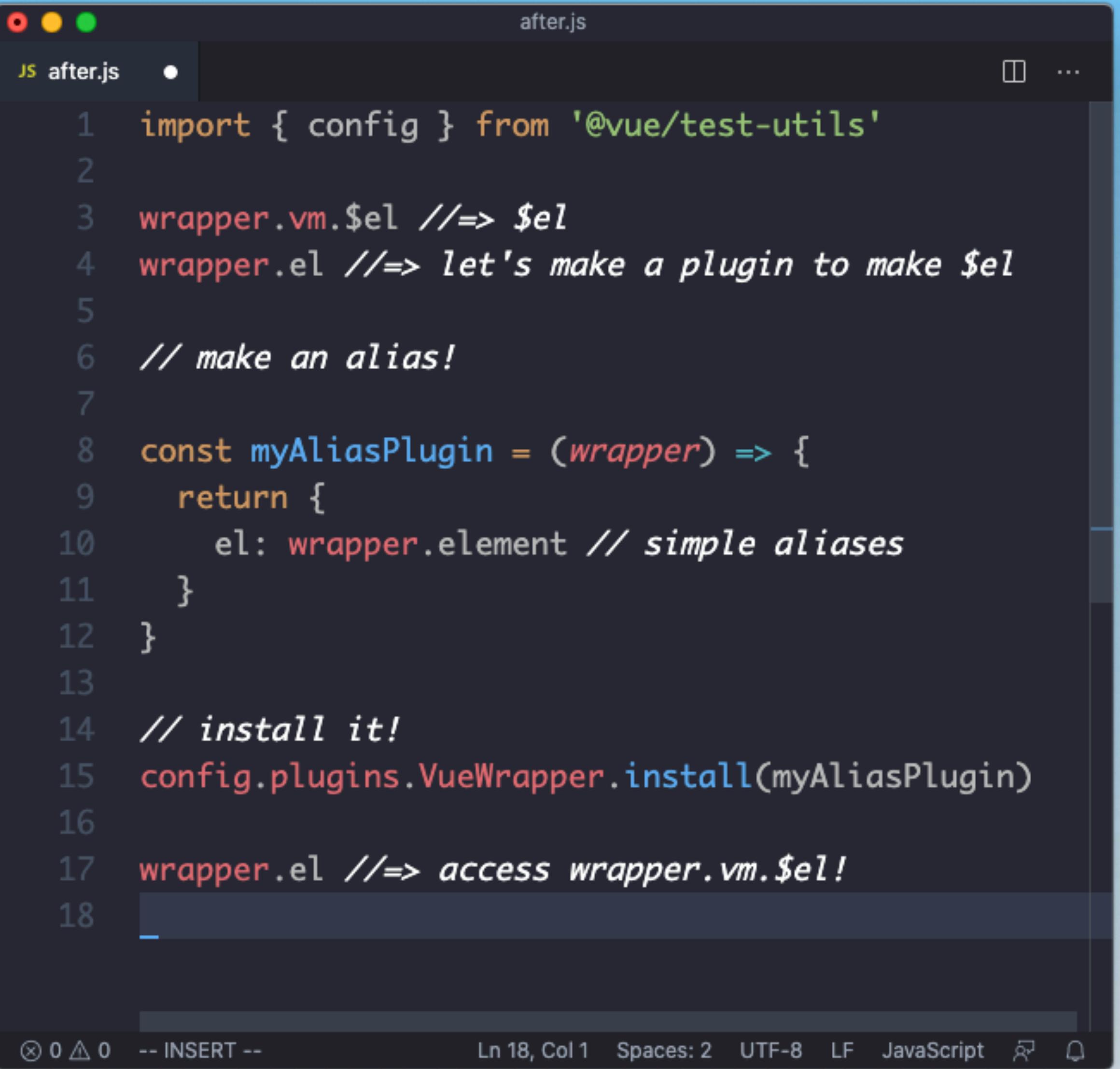
**Before:**

```
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

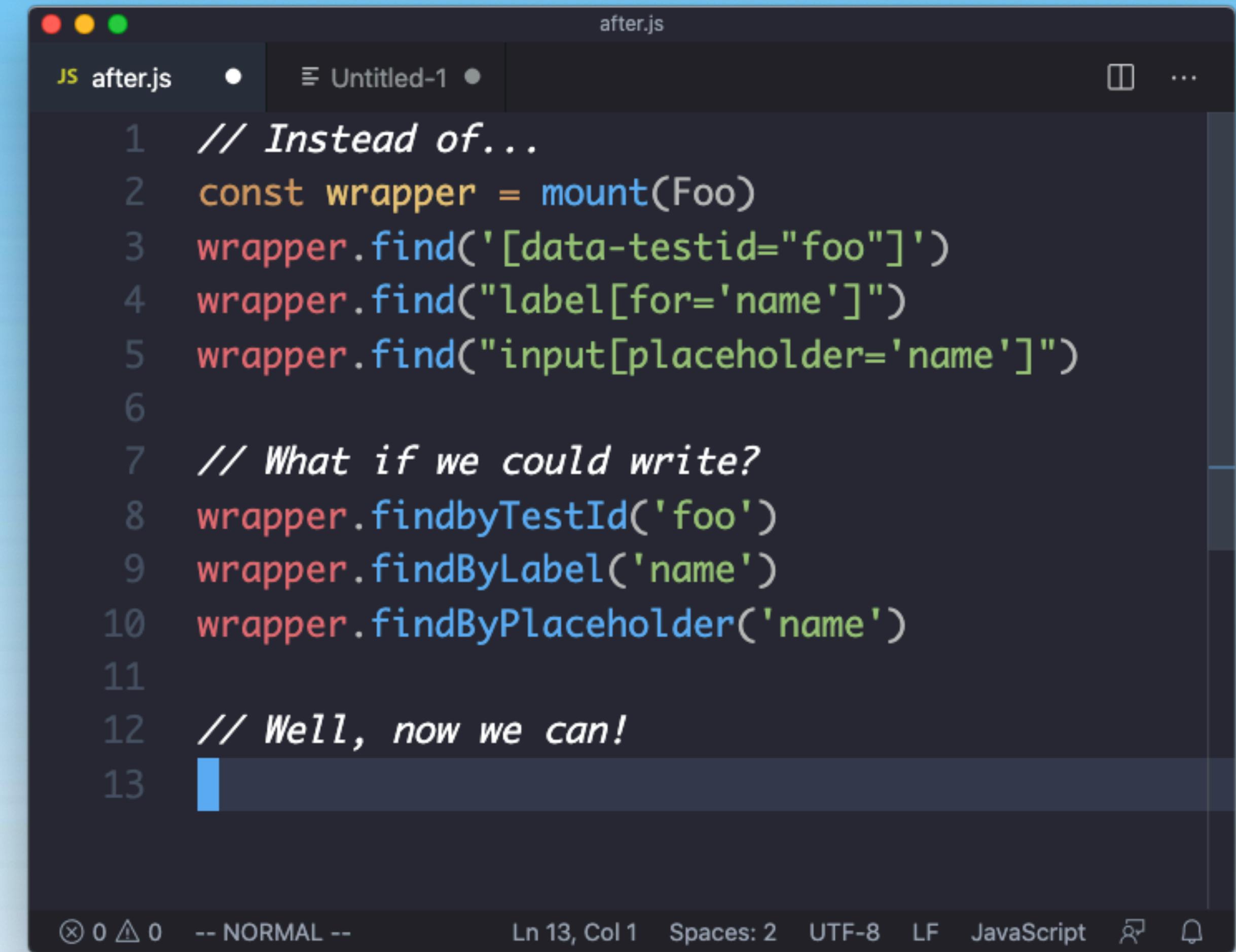


```
JS after.js ● after.js
1 import { config } from '@vue/test-utils'
2
3 wrapper.vm.$el //=> $el
4 wrapper.el //=> let's make a plugin to make $el
5
6 // make an alias!
7
8 const myAliasPlugin = (wrapper) => {
9   return {
10     el: wrapper.element // simple aliases
11   }
12 }
13
14 // install it!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

The screenshot shows a code editor window titled "after.js". The code demonstrates how to create a plugin for Vue Test Utils. It imports the config module from '@vue/test-utils'. It then defines a plugin function that takes a wrapper as an argument and returns an object with an "el" key set to the wrapper's element. This allows the alias "\$el" to be used instead of the original "el" property. The code also shows how to install this plugin using the config.plugins.VueWrapper.install method. Finally, it demonstrates that the alias is working correctly by accessing "wrapper.el". The code editor interface includes tabs for JS and after.js, a status bar at the bottom, and a dark theme.

# New Feature: Plugins!

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



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

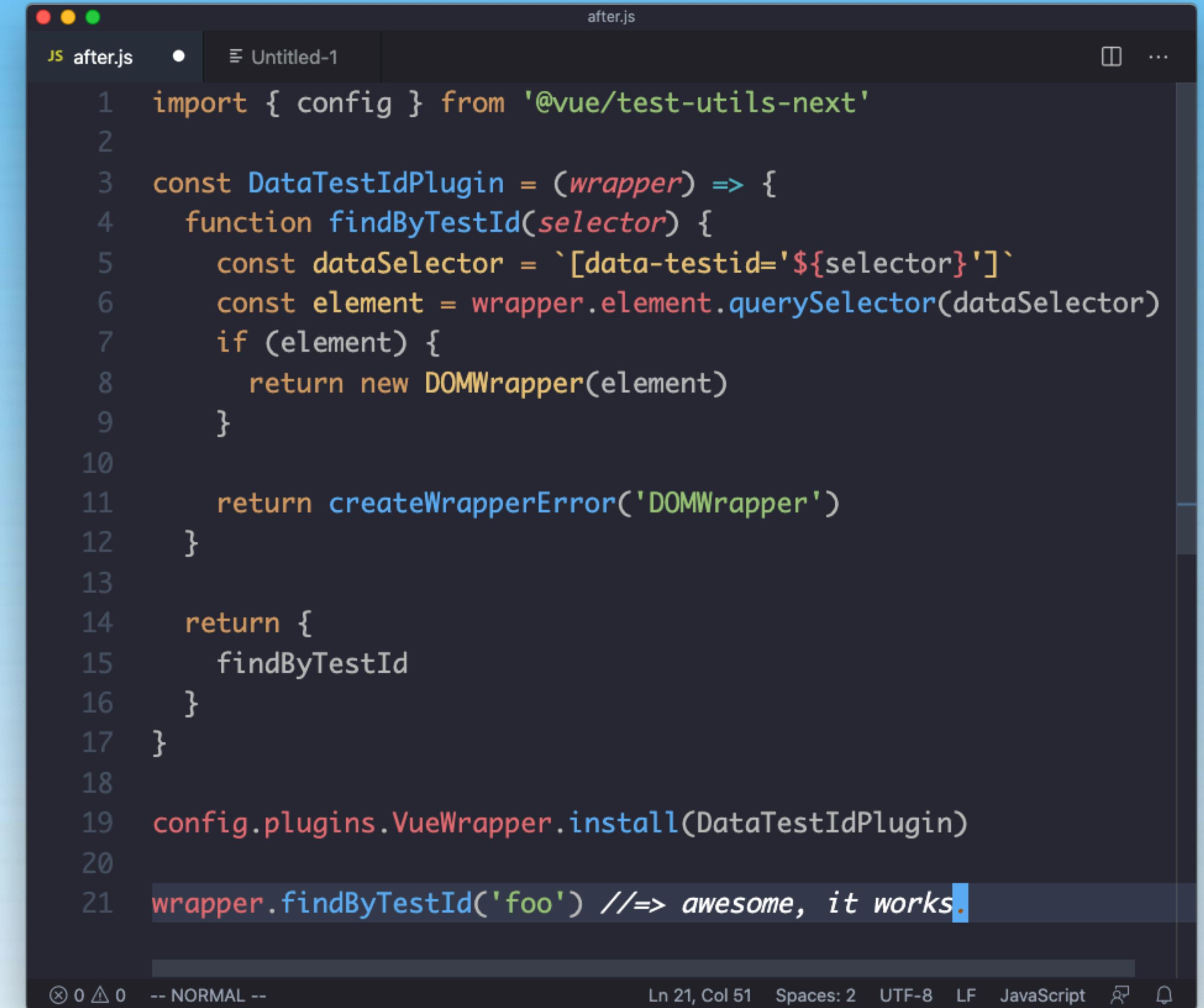
// What if we could write?
wrapper.findByTestId('foo')
wrapper.findByLabel('name')
wrapper.findByPlaceholder('name')

// Well, now we can!
```

The screenshot shows a code editor window titled "after.js". The code demonstrates the addition of new methods to the wrapper object for testing. It starts with a comment indicating the removal of existing methods like `find` and replaces them with new methods: `findByTestId`, `findByLabel`, and `findByPlaceholder`. The code editor interface includes tabs for "JS after.js" and "Untitled-1", status bar indicators for line count (13), column count (1), and file type (JavaScript), and a bottom right corner with icons for search, refresh, and notifications.

# 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", "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, highlighting gotchas. It includes a code snippet for a "PostList" component using Axios to fetch posts from an API.

Making HTTP requests

Modern test runners already provide lots of great features when it comes to test HTTP requests. Thus, Vue Test Utils doesn't feature any unique tool to do so.

However, it is an important feature to test, and there are a few gotchas we want to highlight.

In this section, we explore some patterns to perform, mock, and assert HTTP requests.

## # A list of blog posts

Let's start with a basic use case. The following `PostList` component renders a list of blog posts fetched from an external API. To get these posts, the component features a `button` element that triggers the request:

```
<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

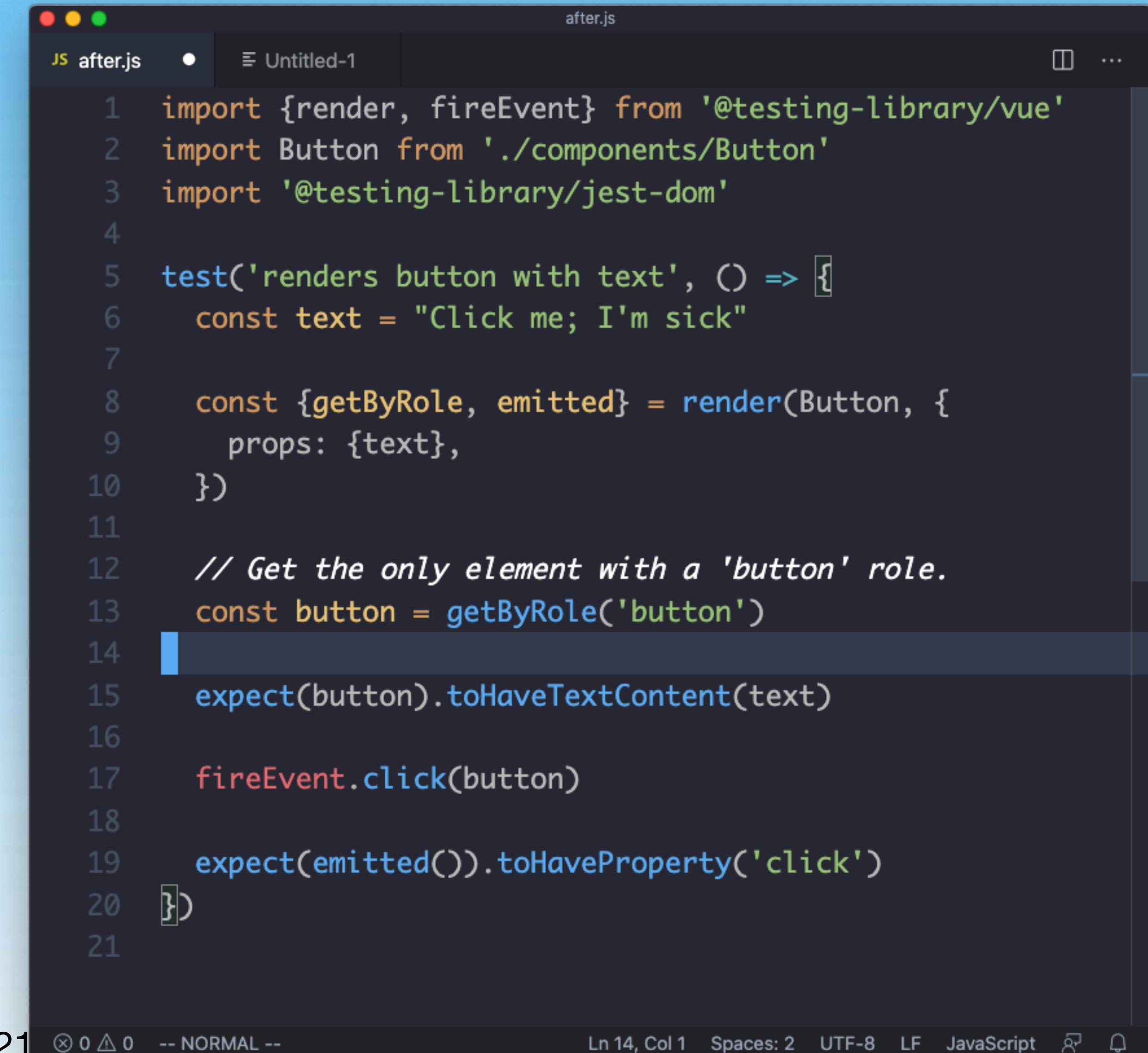
Simple and complete testing utilities that encourage good testing practices



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>

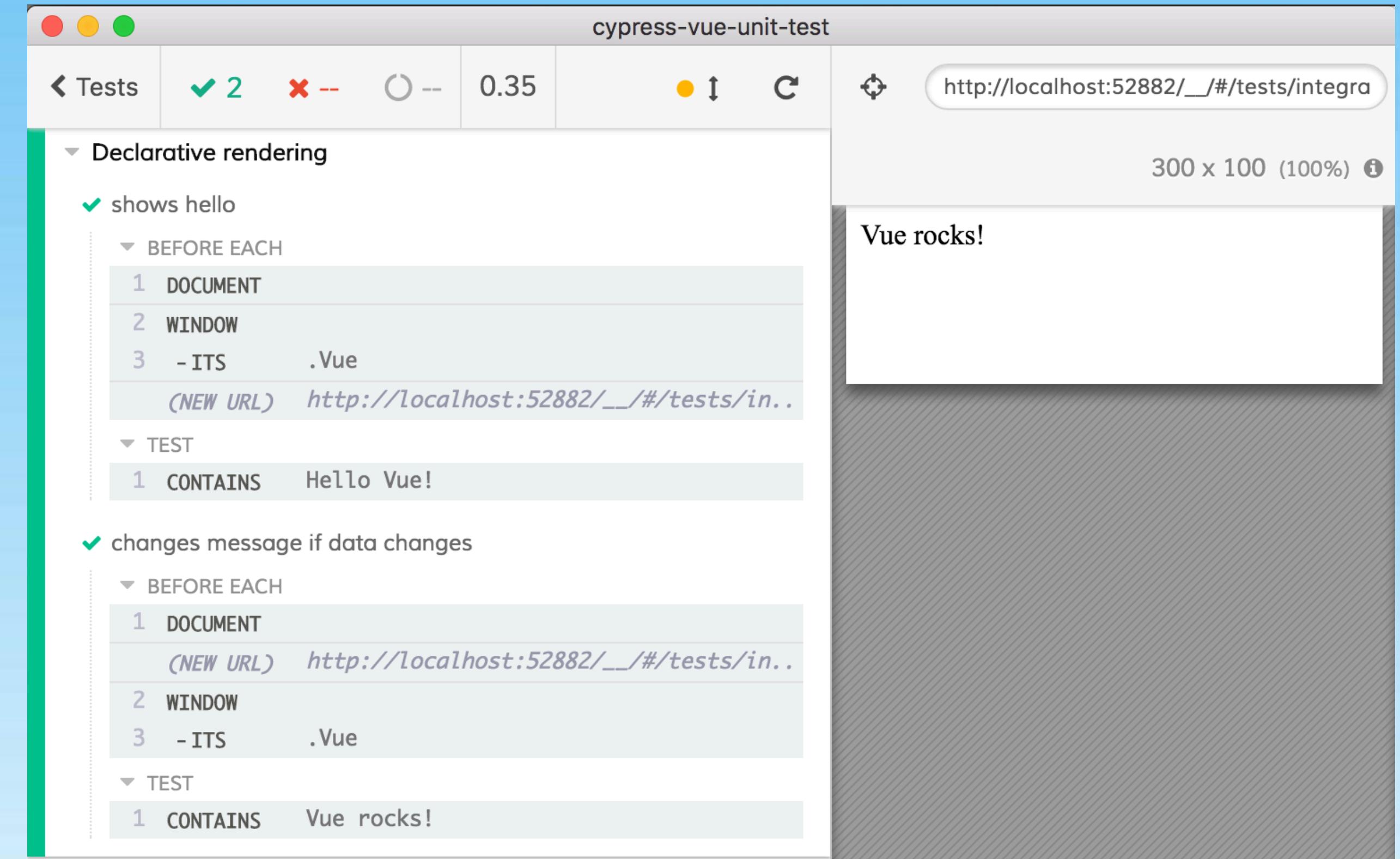


The screenshot shows a code editor window titled 'after.js'. The code is a Jest test for a Vue component. It imports rendering utilities from '@testing-library/vue', the component under test from './components/Button', and Jest's DOM testing library. The test checks if the button renders with the correct text ('Click me; I'm sick'). It then simulates a click on the button and expects an 'click' event to be emitted.

```
JS after.js 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
```

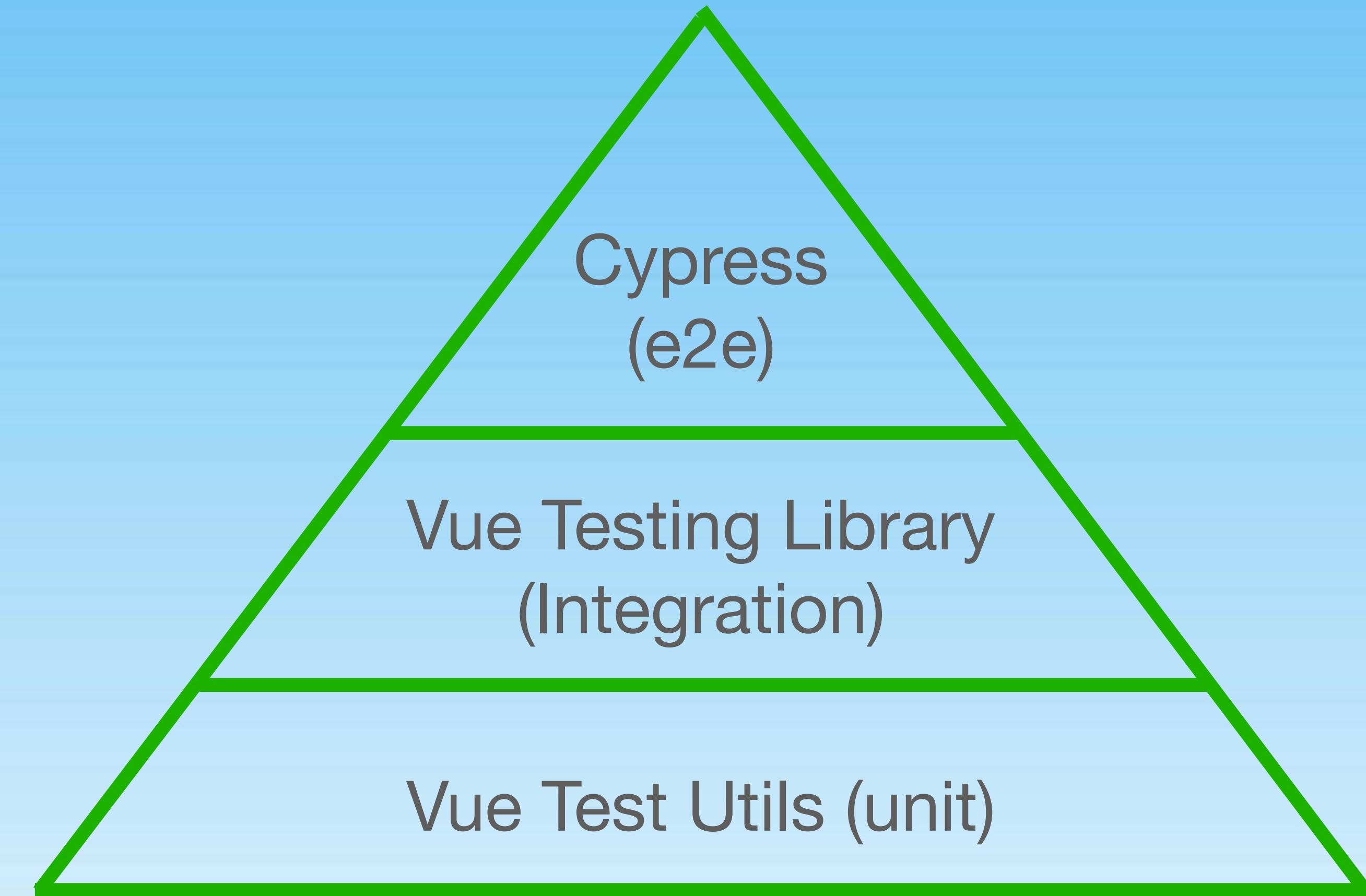
# Case Study: Cypress

- 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://github.com/bahmutov/cypress-vue-unit-test>



# 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 browser window 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 an overview of the documentation: "This is the documentation for Vue Test Utils v2, which targets Vue 3." Below this, there's a section titled "What Next?" with two bullet points: "Vue Test Utils 1 targets Vue 2." and "Vue Test Utils 2 targets Vue 3.". A "What is Vue Test Utils?" section follows, explaining that VTU is a set of utility functions for testing Vue.js components. An example code snippet is shown:

```
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)". A large blue box on the left contains the text "Vue.js", "The Composition API", and "Hi! Get \$10 off my new course on Vue.js 3, TypeScript and, testing, with the discount code VUEJS\_COURSE\_10\_OFF.". To the right, there's a sidebar with a "Welcome" section containing links to "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 heading "# What is this guide?" followed by a welcome message and a detailed description of the handbook's purpose and structure.

This book is written for Vue.js 3 and Vue Test Utils v2.  
Find the Vue.js 2 version [here](#).

## # What is this guide?

Welcome to the Vue.js testing handbook!

This is a collection of short, focused examples on how to test Vue components. It uses `vue-test-utils`, the official library for testing Vue components, and Jest, a modern testing framework. It covers the `vue-test-utils` API, as well as best practises for testing components.

Each section is independent from the others. We start of by setting up an environment with `vue-cli` and writing a simple test. Next, two ways to render a component are discussed - `mount` and `shallowMount`. The differences will be demonstrated and explained.

From then on, we cover how to test various scenarios that arise when testing components, such as testing components that:

- receive props
- use computed properties
- render other components
- emit events

and so forth. We then move on to more interesting cases, such as:

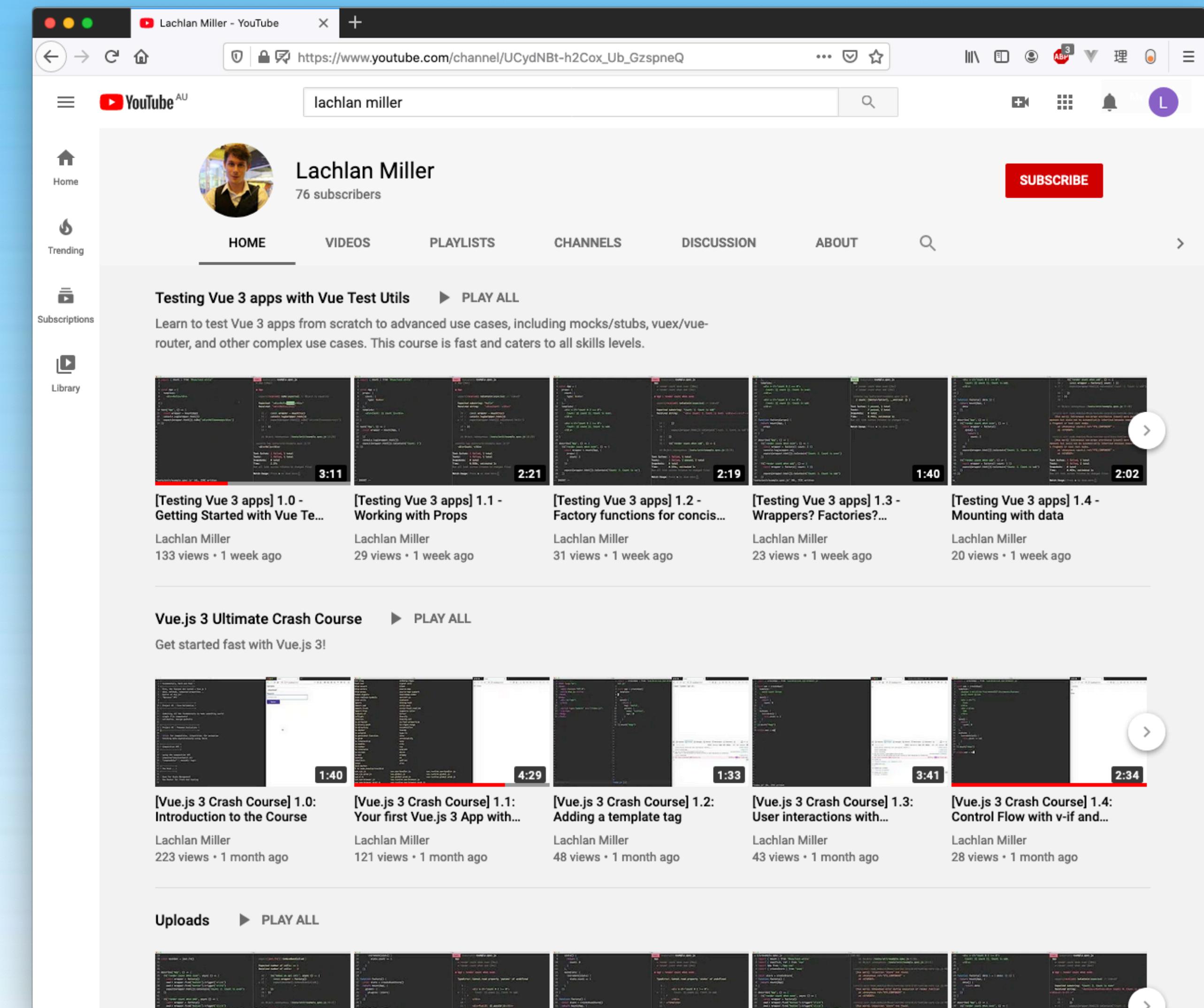
- best practises for testing Vuex (in components, and independently)
- testing Vue router
- testing involving third party components

We will also explore how to use the Jest API to make our tests more robust, such as:

- mocking API responses
- mocking and spying on modules
- using snapshots

# Where to next?

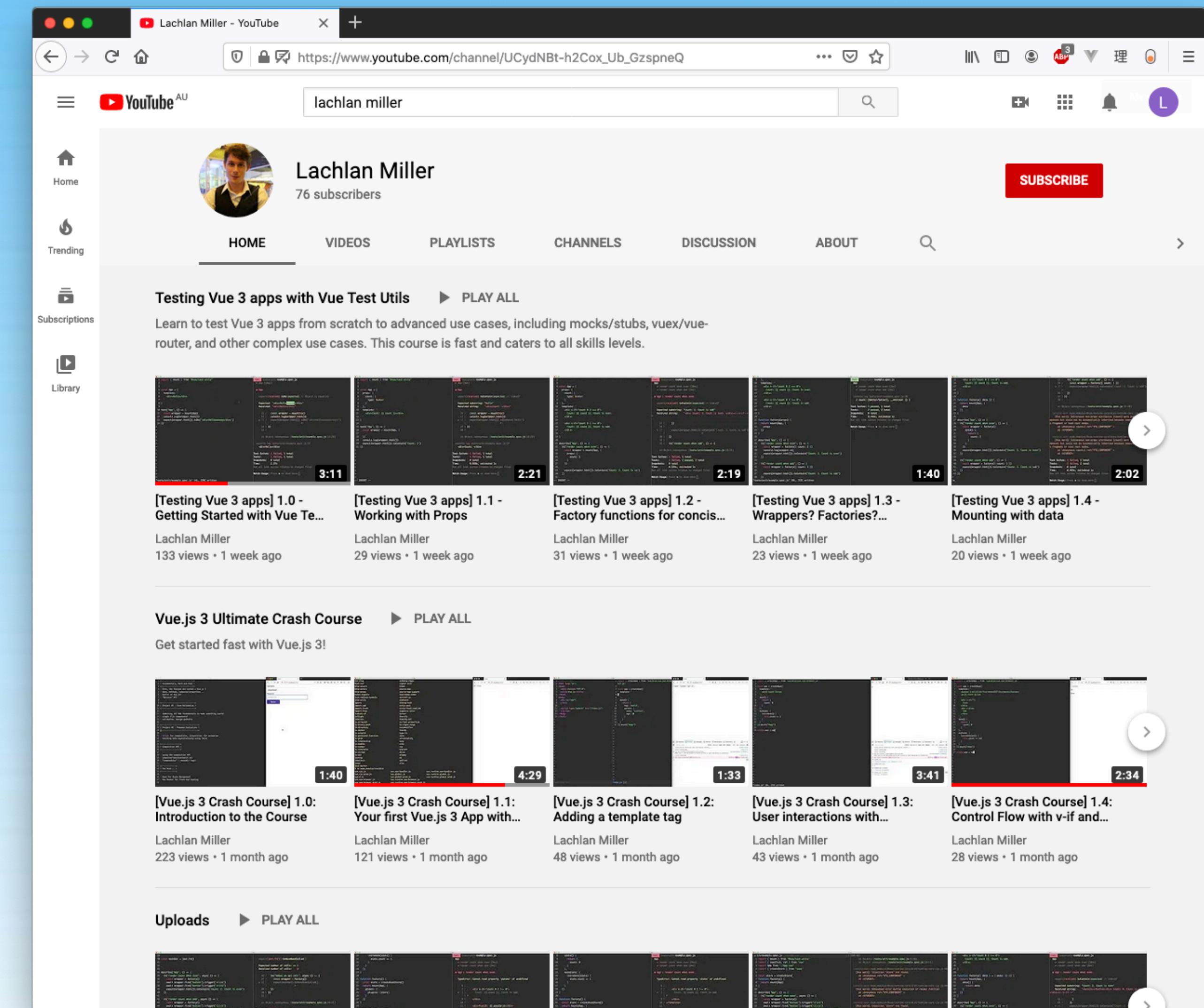
- Video lectures on YouTube
- <https://www.youtube.com/watch?v=0IV4dVYOyBw&list=PLC2LZCNWKL9ahK1loODqYxKu5aA9T5IOA>



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

# Where to next?

- Video lectures on YouTube
- <https://www.youtube.com/watch?v=0IV4dVYOyBw&list=PLC2LZCNWKL9ahK1loODqYxKu5aA9T5IOA>



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