

Testing Vue

Some testing will happen now

Motivation

Automated tests help you and your team build **complex Vue applications quickly** and **confidently** by preventing regressions and encouraging you to break apart your application into testable functions, modules, classes, and components.



When to start?

Start testing early! We recommend you begin writing tests as soon as you can.

The longer you wait to add tests to your application, the more dependencies it will have, and the harder it will be to start.

Credits: <https://vuejs.org/guide/scaling-up/testing.html#why-test>

Types

- Unit: Checks that inputs to a given function, class, or composable are producing the expected output or side effects.
- Component: Checks that your component mounts, renders, can be interacted with, and behaves as expected. These tests import more code than unit tests, are more complex, and require more time to execute.
- End-to-end: Checks features that span multiple pages and make real network requests against your production-built Vue application. These tests often involve standing up a database or other backend.

Recommendation

- Unit:
 - Vitest (minimal effort, blazing fast)
 - Jest
 - Peeky
- Component:
 - Vitest (if headless)
 - Cypress
 - Nightwatch v2 (Vue Component testing support)
- End-to-end:
 - Cypress
 - Playwright
 - Nightwatch v2 (with Selenium WebDriver)

Vite

Vite's value proposition has become **so broad** that it's hard to pin down.

To summarize: Vite is a website bundler that can handle your JavaScript, your CSS, your static assets, and just about anything you load into an HTML document.

Vitest

Vitest is a Vi-testing framework built on top of Vite with an eye for both speed and minimal config.

Vitest

Vite Powered

Reuse Vite's config, transformers, resolvers, and plugins - consistent across your app and tests.

Jest Compatible

Expect, snapshot, coverage, and more - migrating from Jest is straightforward.

Smart & instant watch mode

Only rerun the related changes, just like HMR for tests!

ESM, TypeScript, JSX

Out-of-box ESM, TypeScript and JSX support powered by esbuild

Vitest

Features

- ✓ Vite's config, transformers, resolvers, and plugins.
- ✓ Use the same setup from your app to run the tests!
- ✓ Smart & instant watch mode, like HMR for tests!
- ✓ Component testing for Vue, React, Svelte, Lit and more
- ✓ Out-of-the-box TypeScript / JSX support
- ✓ ESM first, top level await
- ✓ Workers multi-threading via **Tinypool**
- ✓ Benchmarking support with **Tinybench**
- ✓ Filtering, timeouts, concurrent for suite and tests
- ✓ **Jest-compatible Snapshot**
- ✓ **Chai** built-in for assertions + **Jest expect** compatible APIs
- ✓ **Tinyspy** built-in for mocking
- ✓ **happy-dom** or **jsdom** for DOM mocking
- ✓ Code coverage via **c8** or **istanbul**
- ✓ Rust-like **in-source testing**

Vitest & Jest

Despite Vitest's sweeping value propositions, its test runner APIs are nearly identical to [Jest](#) across the board: describe, expect, it.each, mock functions, spies, concurrent flags for parallel tests.

Vitest

Spotlight Vitest: add it to a Vite-based Vue project:

Vitest

Spotlight Vitest: add it to a Vite-based Vue project:



```
1 npm install -D vitest happy-dom @testing-library/vue
```

Vitest

Spotlight Vitest: add it to a Vite-based Vue project:

```
1 npm install -D vitest happy-dom @testing-library/vue
```

```
1 // vite.config.js
2 import { defineConfig } from 'vite'
3
4 export default defineConfig({
5   // ...
6   test: {
7     // enable jest-like global test APIs
8     globals: true,
9     // simulate DOM with happy-dom
10    // (requires installing happy-dom as a peer dependency)
11    environment: 'happy-dom'
12  }
13 })
```

Vitest

Spotlight Vitest: add it to a Vite-based Vue project:

```
1 npm install -D vitest happy-dom @testing-library/vue
```

```
1 // vite.config.js
2 import { defineConfig } from 'vite'
3
4 export default defineConfig({
5   // ...
6   test: {
7     // enable jest-like global test APIs
8     globals: true,
9     // simulate DOM with happy-dom
10    // (requires installing happy-dom as a peer dependency)
11    environment: 'happy-dom'
12  }
13 })
```

With TypeScript:

Vitest

Spotlight Vitest: add it to a Vite-based Vue project:

```
1 npm install -D vitest happy-dom @testing-library/vue
```

```
1 // vite.config.js
2 import { defineConfig } from 'vite'
3
4 export default defineConfig({
5   // ...
6   test: {
7     // enable jest-like global test APIs
8     globals: true,
9     // simulate DOM with happy-dom
10    // (requires installing happy-dom as a peer dependency)
11    environment: 'happy-dom'
12  }
13 })
```

With TypeScript:

```
1 // tsconfig.json
2
3 {
4   "compilerOptions": {
5     "types": ["vitest/globals"]
6   }
7 }
```

Vitest

Then create a file ending in `*.test.js` in your project.

You can place all test files in a test directory in project root, or in test directories next to your source files.

Vitest will automatically search for them using the naming convention.

```
1 // MyComponent.test.js
2 import { render } from '@testing-library/vue'
3 import MyComponent from './MyComponent.vue'
4
5 test('it should work', () => {
6   const { getByText } = render(MyComponent, {
7     props: {
8       /* ... */
9     }
10  })
11
12  // assert output
13  getByText('...')
14 })
```



Vitest

package.json:



```
1 {  
2   // ...  
3   "scripts": {  
4     "test": "vitest"  
5   }  
6 }
```

Finally:



```
1 npm test
```

Example

vite.config.ts

```
1 import { fileURLToPath, URL } from 'node:url'
2
3 import { defineConfig } from 'vite'
4 import vue from '@vitejs/plugin-vue'
5
6 // https://vitejs.dev/config/
7 export default defineConfig({
8   plugins: [vue()],
9   resolve: {
10     alias: {
11       '@': fileURLToPath(new URL('./src', import.meta.url))
12     }
13   }
14 })
```

Example

We will be testing a notification bar.

For this purpose please head into
12_testing_vitest and run:



```
1 cd vue-test-project
2 npm install
3 npm run dev
```

Example

In our example we will have to mock the DOM - vitest supports happy-dom and jsdom.

Example

To write our tests, we need to make use of the following common methods, which can be imported from Vitest:

- `describe`: This function accepts a name and a function and is used to group related tests together.
- `test/it`: This function represents the actual block of code that gets tested.
- `expect`: This function is used to test values or create assertions. It accepts an argument `x` that is expected to be an actual value (string, number, object, etc) and evaluates it using any of the supported methods (e.g `toEqual(y)` which checks if `x` is the same as `y`).

Example

Test your progress:



```
1 npm test  
2  
3 and  
4  
5 npm run coverage
```

Example

Let us see the test for error message:

```
1 describe("notification.vue", () => {
2     test("renders the correct style for error", () => {
3         const type = "error";
4         const wrapper = mount(notification, {
5             props: { type },
6         });
7         expect(wrapper.classes()).toEqual(
8             expect.arrayContaining([ "notification--error" ])
9         );
10    });
11 });
```

Please try the same for success & info.

Example

Success message:

```
1  test("renders correct style for success", () => {
2    const type = "success";
3    const wrapper = mount(notification, {
4      props: { type },
5    });
6    expect(wrapper.classes()).toEqual(
7      expect.arrayContaining([ "notification--success" ])
8    );
9  });
```


Example

Info message:



```
1  test("renders correct style for info", () => {
2    const type = "info";
3    const wrapper = mount(notification, {
4      props: { type },
5    });
6    expect(wrapper.classes()).toEqual(
7      expect.arrayContaining(["notification--info"])
8    );
9  });
```

Example

Not empty:

```
1  test("slides down when message is not empty", () => {
2    const message = "success";
3    const wrapper = mount(notification, {
4      props: { message },
5    });
6    expect(wrapper.classes()).toEqual(
7      expect.arrayContaining(["notification--slide"])
8    );
9  });
```

Please try the same for empty.

Example

The emit event misses the correct expect.

```
1 test("emits event when close button is clicked", async() => {
2     const wrapper = mount(notification, {
3         data() {
4             return {
5                 clicked: false,
6             };
7         },
8     });
9     const closeButton = wrapper.find("button");
10    await closeButton.trigger("click");
11    expect("");
12 });
```

You will need the following keywords: **emitted()** and **wrapper.**
and **.toHaveProperty("clear-notification")**

Example

```
1 test("emits event when close button is clicked", async() => {
2     const wrapper = mount(notification, {
3         data() {
4             return {
5                 clicked: false,
6             };
7         },
8     });
9     const closeButton = wrapper.find("button");
10    await closeButton.trigger("click");
11    expect(wrapper.emitted()).toHaveProperty("clear-notification");
12    });
```

You will need the following keywords: **emitted()** and **wrapper.**
and **.toHaveProperty("clear-notification")**

Example

The render event misses the correct except.

```
1 test("renders message when message is not empty", () => {
2     const message = "Something happened, try again";
3     const wrapper = mount(notification, {
4         props: { message },
5     });
6     expect("");
7 });
```

You will need to **find** with the **wrapper** a **p** element which needs **toBe a message**.

Example



```
1 test("renders message when message is not empty", () => {  
2     const message = "Something happened, try again";  
3     const wrapper = mount(notification, {  
4         props: { message },  
5     });  
6     expect(wrapper.find("p").text()).toBe(message);  
7 });
```

Conclusion

Some last words:

- [Cypress](#) is a browser-based test runner and a complementary tool to Vitest. If you'd like to use Cypress, it is suggested using **Vitest for all headless logic** in your application and **Cypress for all browser-based logic**.

End

That was all for this chapter
