Rethinking in Snapshots

In this lesson, we'll be learning more about Snapshots.

WE'LL COVER THE FOLLOWING

Recap

So far, you've seen how you can test Vue.js components' structure, styles, methods, computed properties, events, watchers, and more. And you've learned to do that using different techniques and methods.

But, what if we tell you that you can test most of it using snapshot testing alone?

You've already seen snapshots being used in chapters 1 and 2 but those chapters were more focused on explaining shallow and deep rendering, so we haven't explained them in detail yet.

Snapshot Testing in Details

> **Snapshot testing** is the technique of asserting by comparing two different outputs.

Think of it as something similar to what the screenshot technique used in end-to-end tests to check for regressions: the first test run takes a screenshot of a part of the screen (for instance: a button), and from that moment on, all the following runs of the same test will compare a new screenshot with the

original one. If they're the same, the test passes, otherwise, there is a regression.

Snapshot testing works in the same way but instead of comparing images, it compares serializable output such as json, html or just strings.

Since Vue.js renders html, you can use snapshot testing to assert the rendered html given different states of a component.

Rethinking in Snapshots

For this example, let's consider the following `ContactBox.vue` component:

ContactBox.vue

```
<template>
  <div :class="{ selected: selected }" @click="handleClick">
    <p>{{ fullName }}</p>
  </div>
</template>

<script>
  export default {
    props: ["id", "name", "surname", "selected"],
    computed: {
      fullName() {
        return `${this.name} ${this.surname}`;
      }
    },
    methods: {
      handleClick() {
        this.$emit("contact-click", this.id);
      }
    }
  };
</script>
```

In this case, we can test several things of this component:

- `fullName` is the combination of `name` + `surname`
- It has a `selected` class when the component is selected
- Emits a `contact-click` event

One way to create tests that validate these specifications would be to check everything separately: the classes attached to the DOM elements, the HTML structure, the computed properties, and state.

Power of Snapshot Testing?

As you've seen in other chapters, you could do these tests as follows:

```
import { mount } from "vue-test-utils";
import ContactBox from "../src/components/ContactBox";

const createContactBox = (id, name, surname, selected) =>
  mount(ContactBox, {
    propsData: { id, name, surname, selected }
  });

describe("ContactBox.test.js", () => {
  it("fullName should be the combination of name + surname", () => {
    const cmp = createContactBox(0, "John", "Doe", false);
    expect(cmp.vm.fullName).toBe("John Doe");
  });

  it("should have a selected class when the selected prop is true", () => {
    const cmp = createContactBox(0, "John", "Doe", true);
    expect(cmp.classes()).toContain("selected");
  });

  it("should emit a contact-click event with its id when the component is clicked", () => {
    const cmp = createContactBox(0, "John", "Doe", false);
    cmp.trigger("click");

    const payload = cmp.emitted("contact-click")[0][0];
    expect(payload).toBe(0);
  });
});
```

But now let's think about how snapshot testing can help us here.

If you think about it, the component renders according to its state. Let's call that the **rendering state**.

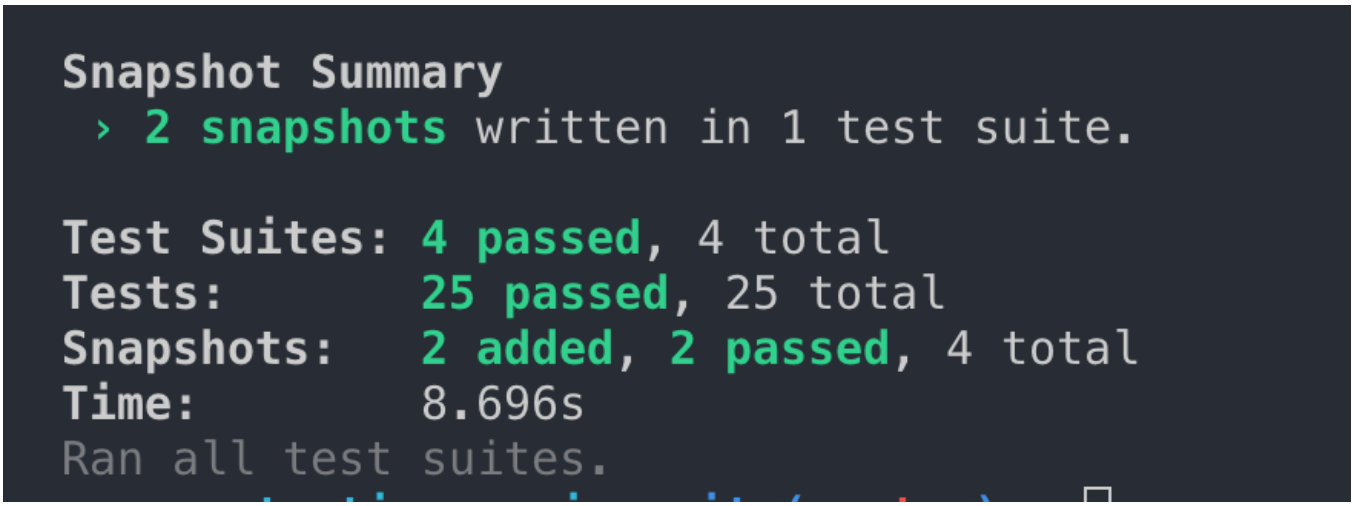
With snapshot testing, instead of worrying about checking for specific things like attributes, classes, methods, computed props and so on, we can instead check the rendering state as it is the projected result of the component state.

For that, you can use snapshot testing for the first test from above as follows:

```
it("fullName should be the combination of name + surname", () => {
  const cmp = createContactBox(0, "John", "Doe", false);
  expect(cmp.element).toMatchSnapshot();
});
```

As you can see, now instead of checking things separately, I'm just asserting the snapshot of `cmp.element`, as it is the rendered HTML of the component.

If you run the test suite now, a `ContactBox.test.js.snap` file should've been created and you'll see a message in the console output as well:



Let's analyze the snapshot generated:

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[
  `ContactBox.test.js fullName should be the combination of name + surname 1`
] = `
<div
  class=""
>
  <p>
    John Doe
  </p>
</div>
`;
```

The purpose of this test is to check that the computed property `fullName` combines both name and surname, separated by a space. Taking a look at the snapshot, you can see that *John Doe* is there, so you can consider this test valid.

In the same way, you can write the second test using snapshot testing:

```
it("should have a selected class when the selected prop is true", () => {
  const cmp = createContactBox(0, "John", "Doe", true);
  expect(cmp.element).toMatchSnapshot();
});
```

Notice that the only thing that changes between this test and the previous one

is that this one is setting the `selected` property to true.

That's the power of snapshot testing: you play with **different states of the components** while just needing to assert the rendering state.

The purpose of this test is to validate that it has a `selected` class when the property is true. Let's run the test suite, and if you check `ContactBox.test.js.snap` again, you'll see that another snapshot has been added:

```
exports[
  `ContactBox.test.js should have a selected class when the selected prop is true 1`
] = `
<div
  class="selected"
>
  <p>
    John Doe
  </p>
</div>
`;
```

The `selected` class is there, as we expected, so we can consider this one also valid.

When does a Snapshot not help? Let's figure that out in the next lesson.
