Custom Events

In this lesson, we'll see several cases for testing Custom Events.

WE'LL COVER THE FOLLOWING

Custom Events

We can test at least two things in Custom Events:

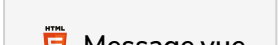
- Asserting that after an action, an event gets triggered
- Checking what an event listener calls when it gets triggered

In the case of the `MessageList.vue` and `Message.vue components` example, that gets translated to:

- Asserting that `Message` components trigger a `message-clicked` when a message gets clicked
- Checking that when a `message-clicked` occurs, a `handleMessageClick` function is called in `MessageList`

Step 1:

First, go to `Message.vue` and use `$emit` to trigger that custom event:



Message.vue

```
<template>
  <li
    style="margin-top: 10px"
    class="message"
    @click="handleClick">
      {{message}}
  </li>
</template>

<script>
export default {
  name: "Message",
  props: ["message"],
  methods: {
    handleClick() {
      this.$emit("message-clicked", this.message)
    }
  }
};
</script>
```

Then, in `MessageList.vue`, handle the event using `@message-clicked`:

MessageList.vue

```
<template>
  <ul>
    <Message
      @message-clicked="handleMessageClick"
      :message="message"
      v-for="message in messages"
      :key="message"/>
  </ul>
</template>

<script>
import Message from "./Message";

export default {
  name: "MessageList",
  props: ["messages"],
  methods: {
    handleMessageClick(message) {
      console.log(message)
    }
  },
  components: {
    Message
  }
};
</script>
```

Step 2:

Now it's time to write a unit test.

Create a nested `describe` within the `test/Message.spec.js` file and prepare the barebones of the test case *"Assert that `Message` components trigger a `message-clicked` when a message gets clicked"* that we mentioned before:

test/Message.spec.js

```
describe("Message.test.js", () => {
  describe("Events", () => {
    beforeEach(() => {
      cmp = createCmp({ message: "Cat" });
    });

    it("calls handleClick when click on message", () => {
      // @TODO
    });
  });
});
```

Testing that the Event Click Calls a Method Handler

The first thing we can test is if the `handleClick` function gets called when clicking a message. For that, we can use a `trigger` of the wrapper component, and a Jest spy using `spyOn` function:

```
it("calls handleClick when click on message", () => {
  const spy = spyOn(cmp.vm, "handleClick");
  cmp.update(); // Forces to re-render, applying changes on template

  const el = cmp.find(".message").trigger("click");
  expect(cmp.vm.handleClick).toBeCalled();
});
```

> See the `cmp.update()` ? When we change things that are used in the template (`handleClick` in this case) and we want the template to apply the changes, we need to use the `update` function.

Keep in mind that by using a spy, the original method `handleClick` will be called. You might intentionally want that, but ideally, we want to avoid it and just check that on clicking, the method is indeed called. For that we can use a Jest Mock function:

```
it("calls handleClick when click on message", () => {
  cmp.vm.handleClick = jest.fn();
  cmp.update();


  const el = cmp.find(".message").trigger("click");
  expect(cmp.vm.handleClick).toBeCalled();
});
```

Here we are totally replacing the `handleClick` method, accessible on the vm of the wrapper component returned by the mount function.

We can make it even easier by using the `setMethods` helper that the official tools provide us:

```
it("calls handleClick when click on message", () => {
  const stub = jest.spy();
  cmp.setMethods({ handleClick: stub });
  cmp.update();

  const el = cmp.find(".message").trigger("click");
  expect(stub).toBeCalled();
});
```

Using `setMethods` **is the suggested way** to do it since it is an abstraction that official tools give us in case the Vue internals change.

Testing that the Custom Event `message-clicked` is Emitted

We've tested that the click method calls its handler, but we haven't tested whether the handler emits the `message-clicked` event itself. We can call the `handleClick` method directly, and use a Jest Mock function in combination with the Vue vm `$on` method:

```
it("triggers a message-clicked event when a handleClick method is called", () => {
  const stub = jest.fn();
  cmp.vm.$on("message-clicked", stub);
  cmp.vm.handleClick();

  expect(stub).toBeCalledWith("Cat");
});
```

Notice that here we're using `toBeCalledWith` so we can assert exactly which parameters we expect, making the test even more robust. Note that we're not using `cmp.update()` here, since we're making no changes that need to propagate to the template.

Testing that @message-clicked Triggers an Event

For custom events, we cannot use the `trigger` method, since it's just for DOM events. But, we can emit the event ourselves by getting the Message component and using its `vm.$emit` method. So add the following test to `MessageList.test.js`:

MessageList.test.js

```
it("Calls handleMessageClick when @message-click happens", () => {
  const stub = jest.fn();
  cmp.setMethods({ handleMessageClick: stub });
  cmp.update();

  const el = cmp.find(Message).vm.$emit("message-clicked", "cat");
  expect(stub).toBeCalledWith("cat");
});
```

You can test what `handleMessageClicked` does by yourself.

Wrapping up

In this chapter, we've seen several cases of properties and events being tested. `vue-test-utils`, the official Vue testing tools, makes this much easier indeed.

Let's test a running project of what we have done so far in the next lesson.
