Computed Properties

In this lesson, we will be testing Computed Properties.

WE'LL COVER THE FOLLOWING

Computed properties and watchers are reactive parts of the logic of Vue.js components. They both serve totally different purposes. One is synchronous and the other asynchronous, which makes them behave slightly differently from each other.

In this chapter, we'll test them and see what different cases we can find on the way.

Computed Properties

Computed properties are simple reactive functions that return data in another form. They behave exactly like the language standard `get/set` properties:

```
class X {
  get fullName() {
    return `${this.name} ${this.surname}`;
  }

  set fullName(value) {
    // ...
  }
}
```

In fact, when you are building *class-based* Vue components, you'll write it just like that. If you're using plain objects, it'd be:

```
export default {
  computed: {
    fullName() {
      return `${this.name} ${this.surname}`;
    }
  }
};
```

You can even add the `set` as follows:

```
export default {
  computed: {
    fullName: {
      get() {
        return `${this.name} ${this.surname}`;
      },
      set(value) {
        // ...
      }
    }
  }
};
```

Testing Computed Properties

Testing a computed property is very simple, and sometimes we don't test a computed property exclusively but test it as part of other tests. However, most times, it's good to have a test for it. Whether that computed property is cleaning up an input or combining data, we want to make sure things work as intended. So let's begin.

Step 1:

First of all, create a `Form.vue` component:

```
require('./check-versions')()

process.env.NODE_ENV = 'production'

var ora = require('ora')
var rm = require('rimraf')
var path = require('path')
var chalk = require('chalk')
var webpack = require('webpack')
var config = require('../config')
var webpackConfig = require('./webpack.prod.conf')

var spinner = ora('building for production...')
```

```
var spinner = ora('building for production...')
spinner.start()

rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {
  if (err) throw err
  webpack(webpackConfig, function (err, stats) {
    spinner.stop()
    if (err) throw err
    process.stdout.write(stats.toString({
      colors: true,
      modules: false,
      children: false,
      chunks: false,
      chunkModules: false
    }) + '\n\n')

    console.log(chalk.cyan('  Build complete.\n'))
    console.log(chalk.yellow(
      '  Tip: built files are meant to be served over an HTTP server.\n' +
      '  Opening index.html over file:// won\'t work.\n'
    ))
  })
})
```

It will show an input, and next to it, the same string but reversed.

It's just a simple example, but enough for a test.

Step 2:

Now, add it to `App.vue` and put it after the `MessageList` component. Remember to import it and include it within the `components` component option.

Step 3:

Then, create a `test/Form.test.js` with the usual bare-bones we've used in other tests:

Form.test.js

```
import { shallowMount } from "@vue/test-utils";
import Form from "../src/components/Form";

describe("Form.test.js", () => {
  let cmp;

  beforeEach(() => {
    cmp = shallowMount(Form);
  });
});
```

Step 4:

Step 4:

Now create a test suite with 2 test cases:

```
describe("Properties", () => {
  it("returns the string in normal order if reversed property is not true", () => {
    cmp.setData({ inputValue: "Yoo" });
    expect(cmp.vm.reversedInput).toBe("Yoo");
  });

  it("returns the reversed string if reversed property is true", () => {
    cmp.setData({ inputValue: "Yoo" });
    cmp.setProps({ reversed: true });
    expect(cmp.vm.reversedInput).toBe("ooY");
  });
});
```

```
require('./check-versions')()

process.env.NODE_ENV = 'production'

var ora = require('ora')
var rm = require('rimraf')
var path = require('path')
var chalk = require('chalk')
var webpack = require('webpack')
var config = require('../config')
var webpackConfig = require('./webpack.prod.conf')

var spinner = ora('building for production...')
spinner.start()

rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {
  if (err) throw err
  webpack(webpackConfig, function (err, stats) {
    spinner.stop()
    if (err) throw err
    process.stdout.write(stats.toString({
      colors: true,
      modules: false,
      children: false,
      chunks: false,
      chunkModules: false
    }) + '\n\n')

    console.log(chalk.cyan('  Build complete.\n'))
    console.log(chalk.yellow(
      '  Tip: built files are meant to be served over an HTTP server.\n' +
      '  Opening index.html over file:// won\'t work.\n'
    ))
  })
})
```

We can access the component instance within `cmp.vm`, so we can access the internal state, computed properties, and methods. Then, testing it is just about changing the value and making sure it returns false when the same string is reversed.

For the second case, it would almost be the same with the difference being that we must set the `reversed` property to true. We could navigate through `cmp.vm…` to change it, but `vue-test-utils` give us a helper method `setProps({ property: value, ... })` that makes it very easy.

That's pretty much it. Depending on the computed property, we may need to define more test cases.

In the next lesson, we will study Watchers.