Rakamin Academy

Core Initiative

Virtual Internship Experience

Applying Jest

Pengenalan mengenai penerapan unit testing menggunakan Jest

Daftar Isi

A. Dasar-Dasar Jest

Dalam melakukan testing menggunakan Jest, format yang didukung adalah *.spec.js atau *.test.js. anda dapat meletakan file tersebut dalam folder test agar memudahkan Jest ketika melakukan testing. Untuk melakukan testing anda dapat mengetikkan syntax "**npm run test:unit**" pada command prompt atau terminal anda. Ada pola umum yang dapat anda gunakan ketika melakukan testing.

```
describe('nama_component', () => {

  // Test
  test('nama_test', () => {

    // Arrange
    const component = {click : () => {}};

    // Act
    component.click();

    // Assert
    expect(component).toBe();
  })

  // Test Lain
})
```

Ada tiga tindakan yang dapat anda lakukan ketika melakukan sebuah test yaitu ***Arrange*** (melakukan konfigurasi dan setup mengenai test yang akan dilakukan), ***Act*** (melakukan sesuatu untuk mendapatkan hasil) dan ***Assert*** (memeriksa apakah hasil sesuai).

B. Testing dengan Jest

Contoh simple dari penerapan testing pada vue akan menggunakan kasus berikut:

Kasus :

Membuat component yang bertujuan untuk melakukan perhitungan dengan tiga opsi perubahan (menambahkan, mengurangi dan mengulang perhitungan).

Test :

Ada beberapa test yang perlu dilakukan untuk memastikan component tadi berjalan sesuai dengan tugasnya :

- Nilai default dari number haruslah 0
- Ketika tombol “add” ditekan maka nilai number akan bertambah
- Ketika tombol “substract” ditekan maka nilai number akan berkurang.
- Ketika tombol “reset” ditekan maka nilai number akan kembali ke nilai default.

Eksekusi :

1. Component

   Membuat component untuk melakukan perhitungan seperti berikut.

   Template :

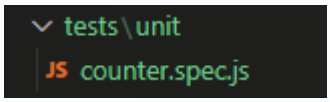
```
<template>

  <div>
    <h1>Counter : </h1>
    <span data-test="number">{{number}}</span>
    <button data-test="add_number" @click="addNumber">Add</button>
    <button data-test="substract_number" @click="substractNumber">Substact</button>
    <button data-test="reset_number" @click="resetNumber">Reset</button>
  </div>

</template>
```

   Data :

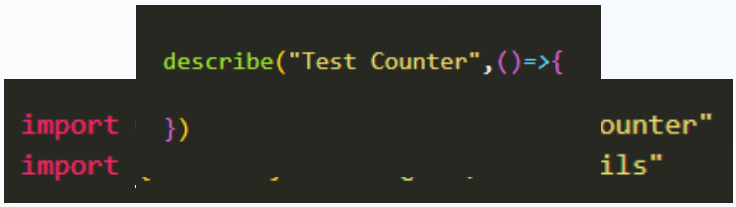
```
methods : {
  addNumber(){
    this.number ++
  },
  substractNumber(){
    this.number--
  },
  resetNumber(){
    this.number = 0
  }
}
```

   Methods :

2. Setup Up Testing
   - Melakukan setup dengan membuat file dengan nama counter.spec.js / counter.test.js pada folder **test**>u**nit.**

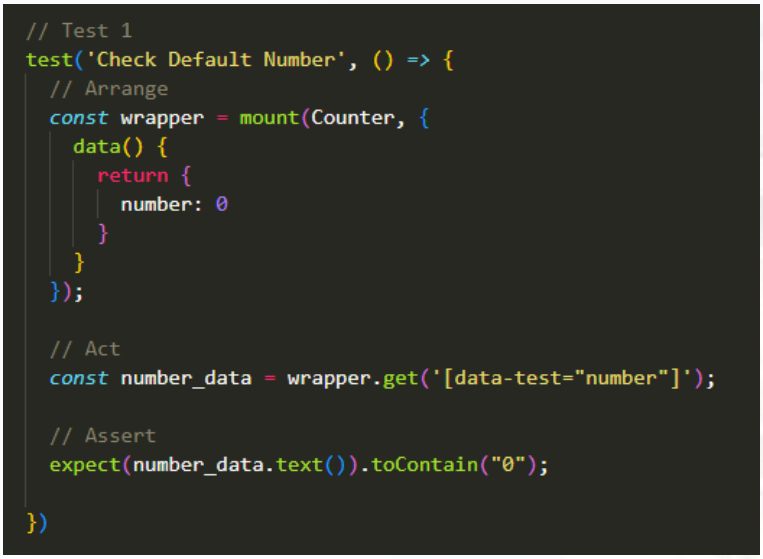


- Mengimport kebutuhan testing seperti component Counter dan fitur mount. Mount sendiri berguna untuk menjalankan component tanpa harus merender component tersebut.
- Membuat describe dari test yang akan dilakukan.



3. Test 1

   Test pertama yang dilakukan adalah dengan memastikan bahwa ketika pertama kali dirender nilai default dari number adalah 0.



   Bagian arrange akan me-mount component counter beserta dengan data awalnya, bagian act akan mengambil tag yang memiliki attribute “data-test=’number’”, lalu bagian assert akan memeriksa apakah isi dari tag yang diambil tadi adalah 0 sesuai dengan default yang ditentukan pada bagian arrange.

4. Test 2

Test kedua yang dilakukan adalah dengan memastikan ketika tombol “add” ditekan maka nilai number akan bertambah senilai 1.

```
// Test 2
test('Add Number with Counter', async () => {
  // Arrange
  const wrapper = mount(Counter, {
    data() {
      return {
        number: 0
      }
    }
  });

  // Act
  const number_data = wrapper.get('[data-test="number"]');
  await wrapper.get('[data-test="add_number"]').trigger('click')

  // Assert
  expect(number_data.text()).toContain("1");
})
```

Sama seperti sebelumnya, pada bagian arrange akan me-mount component counter beserta data awalnya, bagian act akan mengambil tag yang memiliki attribute “data-test=’number’” dan melalakukan click dengan cara mentrigger event pada tag button yang memiliki attribute “data-test=’add_number’”, lalu pada bagian assert akan memastikan apakah data yang ditampilkan akan bernila 1 dan tidak lagi sebagai 0.

5. Test 3

   Test kedua yang dilakukan adalah dengan memastikan ketika tombol “substract” ditekan maka nilai number akan berkurang senilai 0.

```
// Test 3
test('Substract Number with Counter', async () => {
  // Assign
  const wrapper = mount(Counter, {
    data() {
      return {
        number: 1
      }
    }
  });

  // Act
  const number_data = wrapper.get('[data-test="number"]');
  await wrapper.get('[data-test="substract_number"]').trigger('click')

  // Assert
  expect(number_data.text()).toContain("0");
})
```

Masih sama dengan test sebelumnya, yang membedakan adalah data yang diset pada bagain assign adalah 1, tombol yang ditrigger adalah tag yang berattribute “data-test=’substract_number’” dan hasil yang diharapkan pada assert menjadi 0.

6. Test 4

   Test kedua yang dilakukan adalah dengan memastikan ketika tombol “reset” ditekan maka nilai number akan kembali ke nilai 0.

```
// Test 4
test('Reset Number with Counter', async () => {
  const wrapper = mount(Counter, {
    data() {
      return {
        number: 5
      }
    }
  });

  const number_data = wrapper.get('[data-test="number"]');
  await wrapper.get('[data-test="reset_number"]').trigger('click')

  expect(number_data.text()).toContain("0");
})
```

   Masih sama dengan test sebelumnya, yang membedakan adalah data yang diset pada bagain assign adalah 5 (angka berapapun asal tidak 0), tombol yang ditrigger adalah tag yang berattribute “data-test=reset_number’” dan hasil yang diharapkan pada assert menjadi 0.

Apabila testing berhasil maka akan tampil seperti berikut :

```
PASS tests/unit/counter.spec.js
  Test Counter
    √ Check Default Number (41 ms)
    √ Add Number with Counter (31 ms)
    √ Substract Number with Counter (9 ms)
    √ Reset Number with Counter (9 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        3.69 s
Ran all test suites.
```

Catatan :

Tidak semua bagian pada component perlu untuk dilakukan testing. Lakukan testingpada bagian yang memiliki dampak besar ketika terjadi kesalah ketika masa rendering. Berikut ini bagian yang sekiranya dibutuhkan untuk dilakukan testing :

- Output dari sebuah proses khususnya yang akan dirender.
- Event.
- Pemanggilan function.

Daftar Pustaka

[I] https://www.lambdatest.com/jest

[II] https://www.javatpoint.com/jest-framework

[III] https://appkey.id/pembuatan-website/web-programming/jest-adalah/

[IV] https://medium.com/@anandap/mengenal-unit-testing-dengan-jest-7c58eb963edc

[V] https://code.tutsplus.com/id/tutorials/8-things-that-make-jest-the-best-react-testing-framework--cms-30534

[VI] https://www.showwcase.com/show/16345/unit-testing-menggunakan-jest-dan-react-testing-library-di-reactjs