

Tests unitaires

Animé par Mazen Gharbi

Vue & Tests

▷ En Vue, 2 outils sont disponibles pour tester votre application web

- › Mocha
- › Jest



▷ Pour faciliter le unit-testing, Vue met à disposition un utilitaire :

```
.> npm install --save-dev jest @vue/test-utils
```

```
.> npm install --save-dev vue-jest
```

Test naïf d'un composant

```
import { mount } from '@vue/test-utils';
import Component from './component';

describe('Component', () => {
  it('is a Vue instance', () => {
    const wrapper = mount(Component);
    expect(wrapper.isVueInstance()).toBeTruthy();
  });

  it('renders a div', () => {
    const wrapper = mount(Component);
    expect(wrapper.contains('div')).toBe(true);
  });

  it('renders a div', () => {
    const wrapper = mount(Component, {
      propsData: {
        color: 'red'
      }
    });
    expect(wrapper.props().color).toBe('red');
  });
});
```

Permet de monter artificiellement un composant

Plusieurs fonctions fournies

[Liste des fonctions](#)

Intitulé du test

```
it('Test de la fonction du meilleur cours', () => {  
  expect(quelEstLeMeilleurCours()).toBe('ReactJS');  
});
```

▷ **expect** prend une valeur, et en association à **toBe**, il permet de vérifier que la valeur est bien égale au résultat attendu

```
test('valeurs numériques', () => {  
  expect(100).toBeWithinRange(90, 110);  
  expect(101).not.toBeWithinRange(0, 100);  
  expect({ apples: 6, bananas: 3 }).toEqual({  
    apples: expect.toBeWithinRange(1, 10),  
    bananas: expect.not.toBeWithinRange(11, 20),  
  });  
});
```

Thématique de test

▷ Il est possible de définir des thématiques de test

```
describe('Les différentes synthaxes en JEST', () => {  
  it('Premier test simple', () => {  
    expect(1 + 2).toEqual(3);  
    expect(2 + 2).toEqual(4);  
  });  
  
  it('Simple boolean', () => {  
    expect([1]).toBeTruthy();  
    expect(0).toBeFalsy();  
  });  
  
  it('Manipulation sur objet', () => {  
    const houseForSale = {  
      bath: true,  
      bedrooms: 4  
    };  
  
    expect(houseForSale).toHaveProperty('bath');  
    expect(houseForSale).toHaveProperty('bedrooms', 4);  
    expect(houseForSale).not.toHaveProperty('pool');  
  })  
});
```

Gestion des tests asynchrones

▷ Soit le code suivant :

```
export default function asynchronousRequest() {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => {  
      resolve(true);  
    }, 2000);  
  });  
}
```

Test.js

Test.spec.js

```
it('Tester le retour asynchrone', async () => {  
  expect.assertions(1); // S'attend à 1 appel asynchrone durant ce test  
  await expect(asynchronousRequest()).resolves.toBeTruthy();  
});
```

Problème

✓ Tester le retour asynchrone (2004ms)

Manipuler le temps



- ▷ Un test doit être **F.I.R.S.T**
- ▷ Nos tests doivent être enclenchés à chaque modification du code
 - › Donc ils doivent être rapide !

```
jest.useFakeTimers();

it('Tester le retour asynchrone', async () => {
  expect.assertions(1); // S'attend à 1 appel asynchrone durant ce test

  const promise = asynchronousRequest().then(resolved => {
    expect(resolved).toBeTruthy();
  });

  jest.advanceTimersByTime(2000);
  return promise;
});
```

✓ Tester le retour asynchrone (1ms)

Mock

▷ Un appel au serveur étant coûteux en terme de temps, nous allons **bypasser** le comportement natif d'axios pour **simuler** l'appel

```
import axios from 'axios';
```

user.service.js

```
class Users {  
  static getAllUsers() {  
    return axios.get('/users').then(resp => resp.data);  
  }  
}
```

user.service.test.js

```
jest.mock('axios'); // SURCHARGE !
```

```
it('Récupère les utilisateurs du site', () => {
```

```
  const users = [{name: 'Bob'}];
```

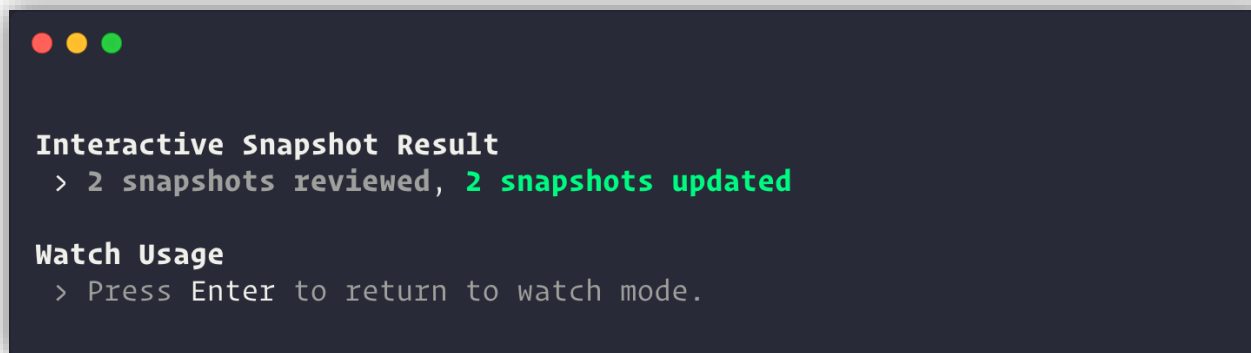
```
  const resp = {data: users};
```

```
  axios.get.mockResolvedValue(resp);
```

```
  return Users.getAllUsers().then(data => expect(data).toEqual(users));  
});
```


Snapshots

- ▷ Jest a introduit une notion de tests « **Snapshot** »
 - › Capture
- ▷ Permet de **figer l'état** d'un composant à un moment **t** et de comparer les différents « snapshot »
 - › Permet de repérer les éventuels différences



```
Interactive Snapshot Result
> 2 snapshots reviewed, 2 snapshots updated

Watch Usage
> Press Enter to return to watch mode.
```

Snapshots

- ▷ Si le snapshot change et que l'on compare les différentes version, le « test snapshot » échouera ;
 - › Ce n'est pas forcément une mauvaise chose !
- ▷ Une fois le « Snapshot test » échoué, vous décidez si la différence doit provoquer un échec du test ou non.
 - › Très utile pour nous assurer que l'interface utilisateur ne change pas de manière inattendue
- ▷ Très pratique pour détecter rapidement les régressions

Snapshots

```
// HelloWorld.spec.js
import { shallowMount } from '@vue/test-utils'
import HelloWorld from './HelloWorld.vue'
```

```
describe('HelloWorld.vue', () => {
  it('renders props.msg when passed', () => {
    const msg = 'new message'
    const wrapper = shallowMount(HelloWorld, {
      propsData: { msg }
    })
    expect(wrapper).toMatchSnapshot()
  })
})
```

Comme « mount » mais sans les enfants



Capture d'écran



Mettre à jour les snapshot

- Comme indiqué précédemment, c'est à nous développeur d'indiquer si la différence entre les 2 snapshot est un « *bug ou une feature* »

Watch Usage

- > Press a to run all tests.
- > Press f to run only failed tests.
- > Press p to filter by a filename regex pattern.
- > Press t to filter by a test name regex pattern.
- > Press u to update failing snapshots.
- > Press i to update failing snapshots interactively.
- > Press q to quit watch mode.
- > Press Enter to trigger a test run.

—

Questions