



Tests Automatisés



▸ Vérification manuelle

- Ecrire une recette de tests et demander à une personne de la rejouer à des étapes clés (nouvelle version)
- Ecrire le test sous la forme de code, et vérifier visuellement que les résultats attendus soit les bons

▸ Tests automatisés

- Le test est codé, la vérification se fait dans un rapport

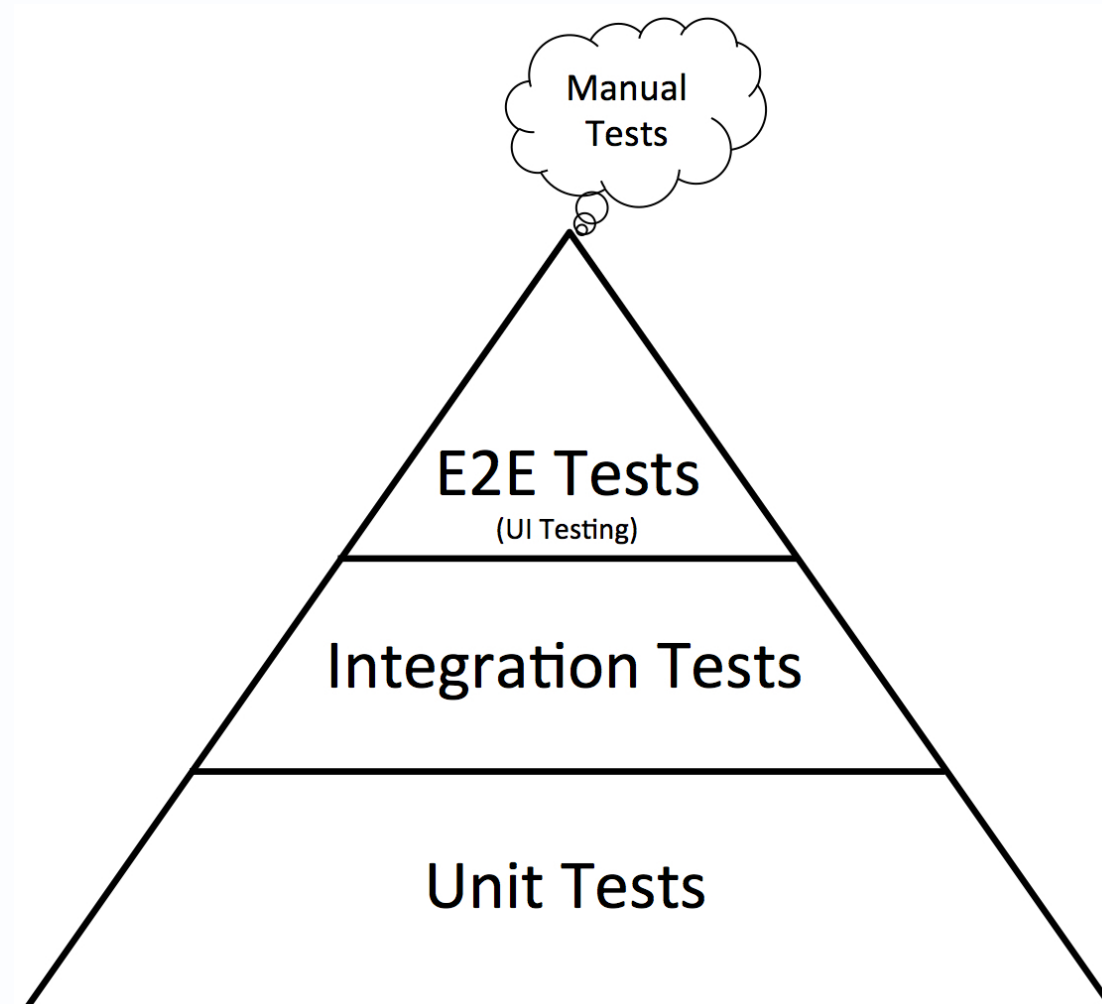
▸ Historique

- sUnit en 1994 (SmallTalk), JUnit en 1997 (Java)
- Les frameworks s'inspirant de jUnit sont catégorisés xUnit (PHPUnit, CUnit...)

Tests automatisés - Pyramide des Tests



- Types de tests de code
 - **Unitaire** : tests de fonction ou de méthodes d'une classe de manière isolée
 - **Intégration** : teste l'intégration entre plusieurs classes
 - **Fonctionnels** : teste l'application du point de vue du client (HTTP dans le cas du web)
 - **End-to-End (E2E)** : teste l'application dans le client (y compris JavaScript, CSS...)



Tests automatisés - Karma



- Lanceur de test

Permet de lancer vos tests simultanément dans Chrome, Firefox, Internet Explorer...

- Installation

`npm install -g karma-cli`

`npm install karma --save-dev`

- Configuration du projet

`karma init`

- Lancement des tests

`karma start`

```
socket.io@1.3.10 (debug@2.1.0, has-binary@0.1.3, socket.io-adapter@0.1.1, sock
Air-de-Romain:Jasmine remain$ karma init

Which testing framework do you want to use ?
Press tab to list possible options. Enter to move to the next question.
> jasmine

Do you want to use Require.js ?
This will add Require.js plugin.
Press tab to list possible options. Enter to move to the next question.
> no

Do you want to capture any browsers automatically ?
Press tab to list possible options. Enter empty string to move to the next question.
> Chrome
> Safari
>

What is the location of your source and test files ?
You can use glob patterns, eg. "js/*.js" or "test/**/*.Spec.js".
Enter empty string to move to the next question.
> █
```

```
Air-de-Romain:Jasmine remain$ karma start
02 09 2015 21:30:11.510:INFO [karma]: Karma v0.13.9 server started at http://localhost:9876/
02 09 2015 21:30:11.518:INFO [launcher]: Starting browser Chrome
02 09 2015 21:30:11.526:INFO [launcher]: Starting browser Safari
02 09 2015 21:30:12.723:INFO [Safari 8.0.7 (Mac OS X 10.10.4)]: Connected on socket HE38slHTBKXL5t5yAAAA with id 54715269
Safari 8.0.7 (Mac OS X 10.10.4): Executed 1 of 1 SUCCESS (0.038 secs / 0.003 secs)
Safari 8.0.7 (Mac OS X 10.10.4): Executed 1 of 1 SUCCESS (0.038 secs / 0.003 secs)
Chrome 45.0.2454 (Mac OS X 10.10.4): Executed 1 of 1 SUCCESS (0.04 secs / 0.008 secs)
TOTAL: 2 SUCCESS
```

Tests automatisés - QUnit



- Créé en 2008 par les développeurs de jQuery
- Type xUnit (JUnit, PHPUnit...) : basés sur des assertions
- Plutôt destiné à du code client
- Installation
 - npm install --save-dev qunitjs
 - bower install --save-dev qunit
- Lancement des tests
 - Ouverture du fichier .html
 - grunt-contrib-qunit
 - karma-qunit

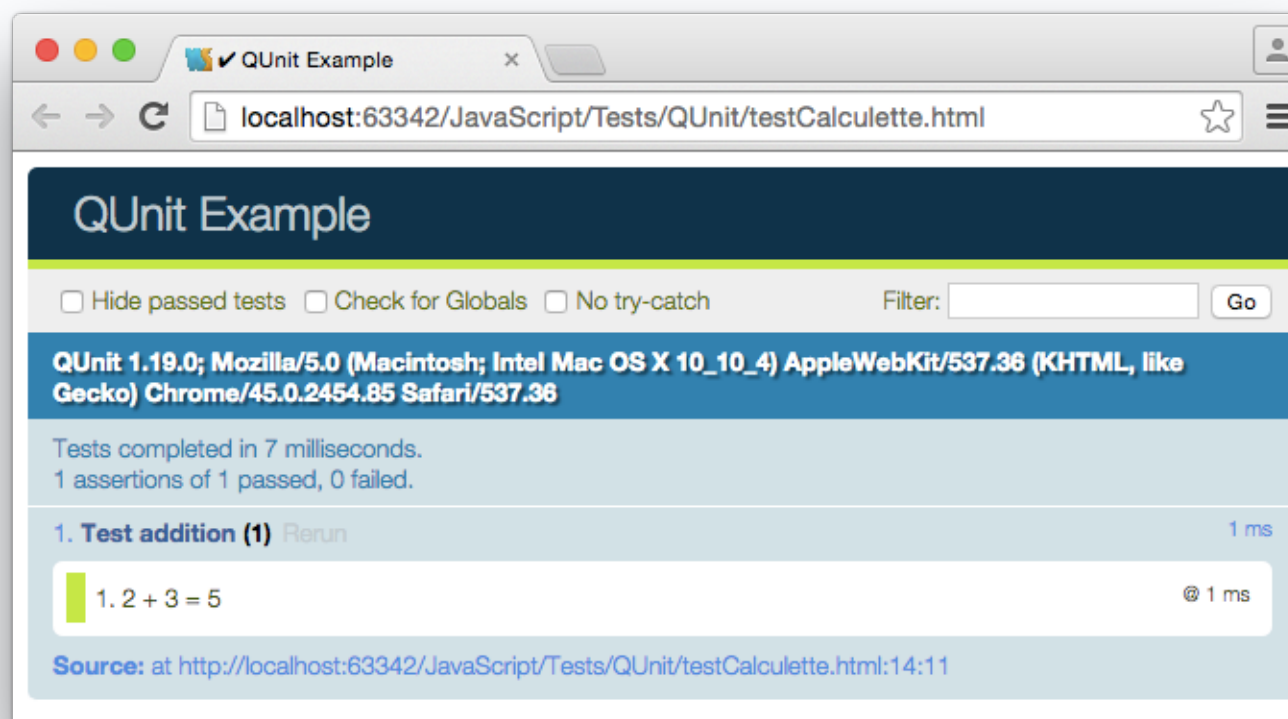


Tests automatisés - QUnit



```
<!-- runner.html -->
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="node_modules/qunitjs/qunit/qunit.css">
</head>
<body>
<div id="qunit"></div>
<div id="qunit-fixture"></div>
<script src="calcullette.js"></script>
<script src="node_modules/qunitjs/qunit/qunit.js"></script>
<script src="calcullette-test.js"></script>
</body>
</html>
```

```
// calcullette-test.js
QUnit.test("Test addition", function(assert) {
  assert.equal(calcullette.ajouter(2, 3), 5, "2 + 3 = 5");
});
```



Tests automatisés - Jasmine



- Créé en 2010
- Type BDD (Behavior-Driven Development)
- Fonctionne pour le browser ou node.js
- Installation et lancement des tests (node)
npm install -g jasmine
jasmine init
jasmine
- Installation et lancement des tests (browser)
npm install --save-dev jasmine-core
SpecRunner.html
karma



Tests automatisés - Jasmine



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Jasmine Spec Runner v2.3.4</title>

  <link rel="shortcut icon" type="image/png" href="node_modules/jasmine-core/images/
jasmine_favicon.png">
  <link rel="stylesheet" href="node_modules/jasmine-core/lib/jasmine-core/jasmine.css">

  <script src="node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
  <script src="node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
  <script src="node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>

  <!-- include source files here... -->
  <script src="calculette.js"></script>

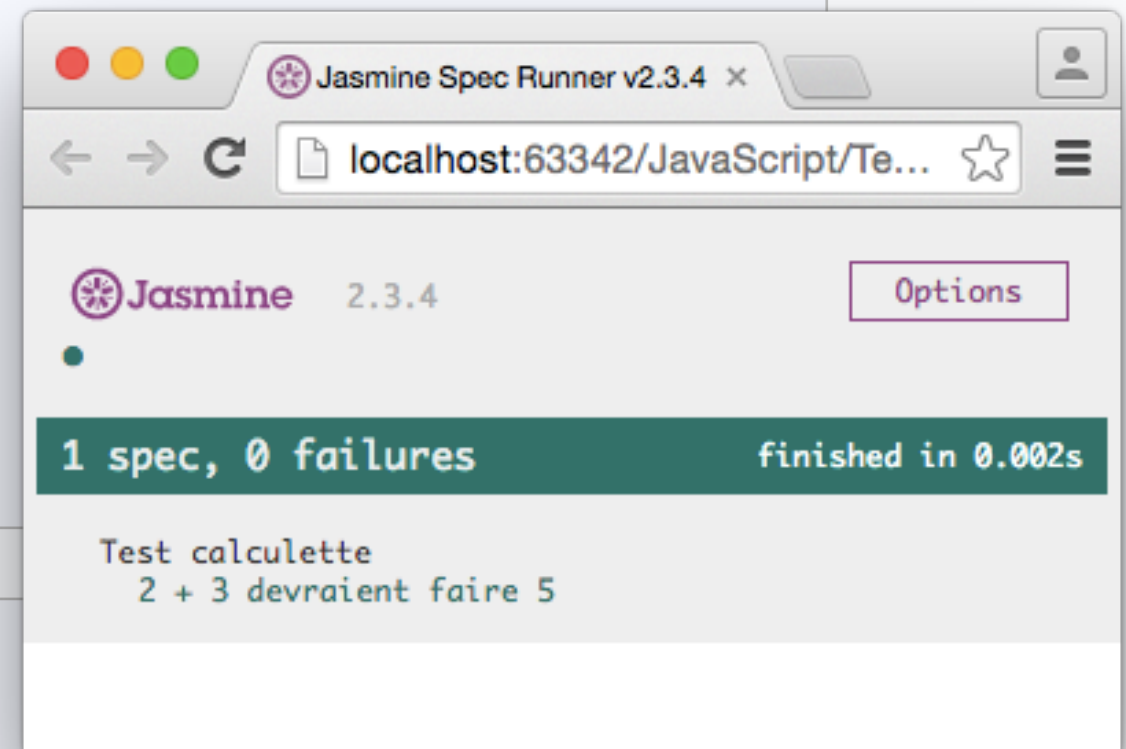
  <!-- include spec files here... -->
  <script src="spec/CalculetteSpec.js"></script>

</head>
<body>
</body>
</html>
```

```
describe("Test calculette", function() {

  it("2 + 3 devraient faire 5", function() {
    expect(calculette.ajouter(2, 3)).toEqual(5);
  });

});
```



Tests automatisés - Mocha



- Créé en 2011
- Type assert ou BDD (le framework est flexible)
- Fonctionne pour le browser ou node.js
- Installation et lancement des tests (node)
npm install -g mocha
mocha
- Installation et lancement des tests (browser)
npm install -g mocha
mocha init
npm install chai
runner.html
karma



Tests automatisés - Mocha



```
<!DOCTYPE html>
<html>
  <head>
    <title>Mocha</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="mocha.css" />
  </head>
  <body>
    <div id="mocha"></div>
    <script src="mocha.js"></script>
    <script src="node_modules/chai/chai.js"></script>
    <script>mocha.setup('bdd');</script>
    <script src="src/calcullette.js"></script>
    <script src="test/calcullette-test.js"></script>
    <script>
      mocha.run();
    </script>
  </body>
</html>
```

```
var assert = chai.assert;

describe('Test Addition', function() {
  it('2 + 3 devraient faire 5', function () {
    assert.equal(5, calcullette.ajouter(2, 3));
  });
});
```



Tests automatisés - Voir aussi



- Coverage :
 - Istanbul : <https://istanbul.js.org>
- Doubles :
 - Sinon : <http://sinonjs.org>
- Parallélisation des tests :
 - Jest : <https://facebook.github.io/jest/>
 - AVA : <https://github.com/avajs/ava>
- Tests End-to-End :
 - Selenium : <http://www.seleniumhq.org>
 - Webdriver : <http://webdriver.io>
 - Cypress : <https://www.cypress.io>
 - CodeceptJS : <https://codecept.io>
- PAAS de tests :
 - Sauce Labs : <https://saucelabs.com>
 - Browser Stack : <https://www.browserstack.com>

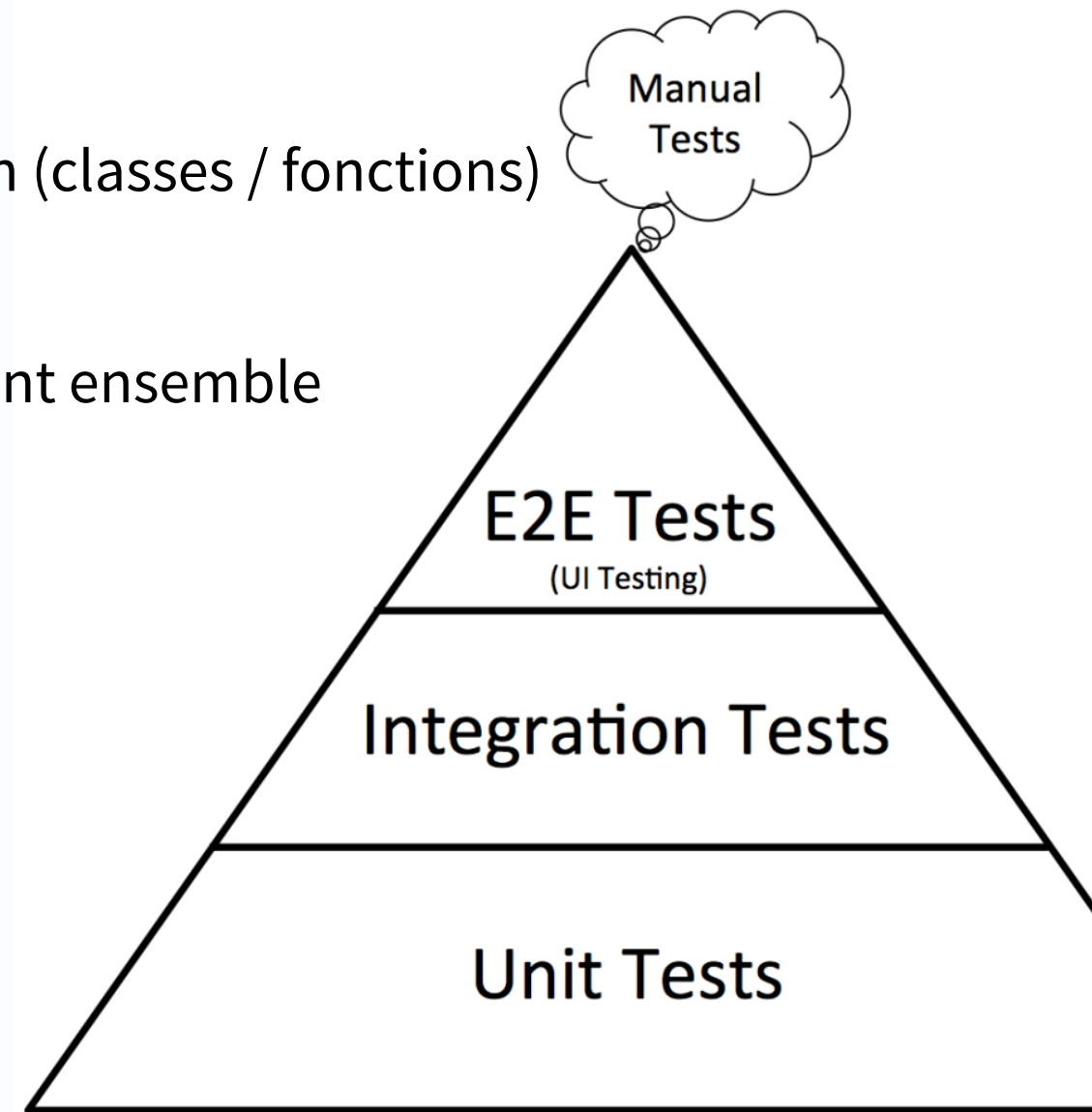


Tests avec Jest

Tests avec Jest - Introduction



- Avec les tests automatisés, les scénarios de tests sont codés et peuvent être rejoués rapidement plus régulièrement.
- 3 types de tests automatisés au niveau code côté Front :
 - Test unitaire
Permet de tester les briques d'une application (classes / fonctions)
 - Test d'intégration
Teste que les briques fonctionnent correctement ensemble
 - Test End-to-End (E2E)
Vérifie l'application dans le client



Tests avec Jest - Introduction



- Framework de test créé en 2014 par Facebook
- Sous Licence MIT depuis septembre 2017
- Permet de lancer des tests :
 - unitaires / d'intégration (dans Node.js)
 - fonctionnels / E2E (via Puppeteer)
- Peut s'utiliser avec ou sans configuration
- Les tests se lancent en parallèle dans les Workers Node.js
- Intègre par défaut :
 - Calcul de coverage (via Istanbul)
 - Mocks (natifs ou en installant Sinon.JS)
 - Snapshots

Tests avec Jest - Installation



- Installation

```
npm install --save-dev jest
```

```
yarn add --dev jest
```
- Déjà intégré à Create React App

Tests avec Jest - Hello, world !



- Sans configuration, les tests doivent se trouver dans un répertoire `__tests__`, ou bien se nommer `*.test.js` ou `*.spec.js`

```
// src/hello.js
const hello = (name = 'World') => `Hello ${name} !`;

module.exports = hello;
```

```
// __tests__/hello.js
const hello = require('../src/hello');

test('Hello, world !', () => {
  expect(hello()).toBe('Hello World !');
  expect(hello('Romain')).toBe('Hello Romain !');
});
```


Tests avec Jest - Lancements des tests



- Si Jest localement
node_modules/.bin/jest
- Si Jest globalement
jest
- Avec un script test dans package.json
npm run test
npm test
npm t

```
// package.json
{
  "devDependencies": {
    "jest": "^22.0.6"
  },
  "scripts": {
    "test": "jest"
  }
}
```

```
MacBook-Pro:hello-jest romain$ node_modules/.bin/jest
```

```
PASS __tests__/hello.js
  ✓ Hello, world ! (3ms)
```

```
Test Suites: 1 passed, 1 total
```

```
Tests: 1 passed, 1 total
```

```
Snapshots: 0 total
```

```
Time: 0.701s, estimated 1s
```

```
Ran all test suites.
```

Tests avec Jest - Watchers



- En mode Watch

```
node_modules/.bin/jest --watchAll
```

```
jest --watchAll
```

```
npm t -- --watchAll
```

```
MacBook-Pro:hello-jest romain$ npm t -- --watchAll
```

```
PASS __tests__/hello.js
```

```
PASS __tests__/calc.js
```

```
Test Suites: 2 passed, 2 total
```

```
Tests: 3 passed, 3 total
```

```
Snapshots: 0 total
```

```
Time: 0.65s, estimated 1s
```

```
Ran all test suites.
```

Watch Usage

- > Press f to run only failed tests.
- > Press o to only run tests related to changed files.
- > Press p to filter by a filename regex pattern.
- > Press t to filter by a test name regex pattern.
- > Press q to quit watch mode.
- > Press Enter to trigger a test run.

Tests avec Jest - Coverage



- Avec calcul du coverage
node_modules/.bin/jest --coverage
jest --coverage
npm t -- --coverage
- Par défaut le coverage s'affiche dans la console et génère des fichiers Clover, JSON et HTML dans le dossier coverage

```
MacBook-Pro:hello-jest romain$ npm t -- --coverage
```

```
PASS __tests__/calc.js  
PASS __tests__/hello.js
```

```
Test Suites: 2 passed, 2 total  
Tests:      3 passed, 3 total  
Snapshots:  0 total  
Time:       0.722s, estimated 1s  
Ran all test suites.
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	86.67	100	60	100	
calc.js	83.33	100	50	100	
hello.js	100	100	100	100	

Tests avec Jest - Mocks



- Jest intègre par défaut une bibliothèque de Mocks

```
// __tests__/Array.prototype.forEach.js
const names = ['Romain', 'Edouard'];

test('Array forEach method', () => {
  const mockCallback = jest.fn();
  names.forEach(mockCallback);
  expect(mockCallback.mock.calls.length).toBe(2);
  expect(mockCallback).toHaveBeenCalledTimes(2);
});
```

Tests avec Jest - Tester les timers



- La fonction `jest.useFakeTimers()` transforme les timers (`setTimeout`, `setInterval`...) en mock

```
// src/timeout.js
const timeout = (delay, arg) => {
  return new Promise((resolve) => {
    setTimeout(resolve, delay, arg);
  });
};

module.exports = timeout;
```

```
// __tests__/timeout.js
jest.useFakeTimers();

const timeout = require('../src/timeout');

test('waits 1 second', () => {
  const arg = timeout(10000, 'Hello');

  expect(setTimeout).toHaveBeenCalledTimes(1);
  expect(setTimeout).toHaveBeenLastCalledWith(expect.any(Function), 10000,
  'Hello');
});
```

Tests avec Jest - React



- Une application créée avec create-react-app est déjà configurée pour fonctionner avec React
- Sinon il faudrait installer des dépendances comme babel, babel-jest...
<https://facebook.github.io/jest/docs/en/tutorial-react.html>

```
// src/App.js
import React, { Component } from 'react';
import { Hello } from './Hello';
import { CounterButton } from './CounterButton';

class App extends Component {
  render() {
    return (
      <div>
        <Hello firstName="Romain" />
        <hr />
        <CounterButton/>
      </div>
    );
  }
}

export default App;
```



- Pour tester un composant React il faut en faire le rendu

```
// src/App.test.js
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

it('renders without crashing', () => {
  const div = document.createElement('div');
  ReactDOM.render(<App />, div);
});
```

- 2 inconvénients ici :
 - Nécessite que document existe (exécuter les tests dans un navigateur ou utiliser des implémentations de document côté Node.js comme JSDOM)
 - Les composants enfants du composant testé seront également rendu, le test n'est pas un test unitaire mais un test d'intégration

Tests avec Jest - Snapshot Testing



- Facebook fourni un paquet npm pour simplifier les tests : react-test-renderer
- Ici on fait un simple Snapshot, c'est à dire une capture du rendu du composant, si lors d'un test futur le rendu est modifié le test échoue

```
import React from 'react';
import renderer from 'react-test-renderer';
import { Hello } from './Hello';

test('it renders like last time', () => {
  const tree = renderer
    .create(<Hello />)
    .toJSON();
  expect(tree).toMatchSnapshot();
});
```


Tests avec Jest - Shallow Rendering



- On peut également faire appel à `ShallowRenderer` qui ne va faire qu'un seul niveau de rendu, et donc rendre le test unitaire

```
// src/App.test.js
import React from 'react';
import App from './App';
import ShallowRenderer from 'react-test-renderer/shallow';
import { Hello } from './Hello';
import { CounterButton } from './CounterButton';

it('renders without crashing', () => {
  const renderer = new ShallowRenderer();
  renderer.render(<App />);
  const result = renderer.getRenderOutput();

  expect(result.type).toBe('div');
  expect(result.props.children).toEqual([
    <Hello firstName="Romain" />,
    <hr />,
    <CounterButton />,
  ]);
});
```

Tests avec Jest - Enzyme



- Facebook recommande également l'utilisation de la bibliothèque Enzyme, créée par AirBnB.
- Elle fournit un API haut niveau (proche de jQuery) pour manipuler les tests des composants

```
import React from 'react';
import { Hello } from './Hello';
import { shallow } from 'enzyme';

test('it renders without crashing with enzyme', () => {
  shallow(<Hello />);
});

test('it renders without crashing with enzyme', () => {
  const wrapper = shallow(<Hello />);
  expect(wrapper.contains(<div>Hello !</div>)).toEqual(true);
});

test('it renders without crashing with enzyme', () => {
  const wrapper = shallow(<Hello firstName="Romain"/>);
  expect(wrapper.contains(<div>Hello Romain !</div>)).toEqual(true);
});
```

Tests avec Jest - Tester des événements



```
import React, { Component } from 'react';

export class CounterButton extends Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState({
      count: this.state.count + 1,
    });
  }

  render() {
    return (
      <button onClick={this.handleClick}>{this.state.count}</button>
    );
  }
}
```

Tests avec Jest - Tester des événements



```
import React from 'react';
import { CounterButton } from './CounterButton';
import { shallow } from 'enzyme';

test('it renders without crashing', () => {
  shallow(<CounterButton />);
});

test('it contains 0 at first rendering', () => {
  const wrapper = shallow(<CounterButton />);
  expect(wrapper.text()).toBe('0');
});

test('it contains 1 after click', () => {
  const wrapper = shallow(<CounterButton />);
  wrapper.simulate('click');
  expect(wrapper.text()).toBe('1');
});
```

Tests avec Jest - Exercices



- Tester unitairement les fonctions liées à Redux :
 - Actions Creators
 - Selectors
 - Reducers
- Tester les composants React (pas les containers) avec Enzyme et les mocks