

Tests unitaires JavaScript avec Jest

Jest est un framework de test JavaScript. Il peut être utilisé pour des projets TypeScript, Node, React, Angular. Dans ce TP nous allons tester un module Node.JS.

Référence

Jest : <https://jestjs.io>

1. Créer le fichier `package.json`

- Ouvrir un terminal, créer un répertoire pour le TP

```
mkdir testjest
cd testjest
```
- Créer le fichier `package.json` dans le répertoire `testjest` (il contient les méta-informations du projet) :

```
npm init -y
```

2. Installer Jest

- Installer localement avec npm :

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

L'option `--save-dev` installe une dépendance de développement (i.e. une dépendance qui est utilisée pendant la phase de développement de projet). La dépendance a été ajoutée dans le fichier `package.json`.

Un répertoire `node_modules` est créé, il contient la commande `jest`.
- Editer le fichier `package.json` pour indiquer le chemin de `jest` dans `node_modules` :

```
"scripts": {
  "test": "node_modules/.bin/jest --coverage --verbose"
}
```

L'option `--coverage` permet d'obtenir la couverture du code.

3. Créer un fichier JavaScript

Créer le fichier `factorielle.js` dans le répertoire `testjest`. Il crée un module Node.js. La fonction factorielle de ce module pourra être importée dans un autre script.

```
function factorielle(n) {
  if ((n == 0) || (n == 1)) {
    return 1;
  } else {
    return n * factorielle(n-1);
  }
}
module.exports = factorielle;
```

4. Créer le fichier de test Jest

Créer le fichier *factorielle.test.js* dans le répertoire *testjest*.

```
const factorielle = require('./factorielle');

test('factorielle(2)', () => {
  expect(factorielle(2)).toBe(2);
});

test('factorielle(3)', () => {
  expect(factorielle(3)).toBe(6);
});

test('factorielle(4)', () => {
  expect(factorielle(4)).toBe(24);
});
```

test est un mot-clé Jest qui prend 2 paramètres :

- Le message affiché par le test (chaîne de caractères)
- Une fonction contenant le code de test.

expect retourne un objet sur lequel un comparateur peut être appliqué.

Dans l'exemple, le comparateur est *toBe()*, il permet de tester une égalité exacte. La négation peut être obtenue avec *.not.toBe()*.

Liste des comparateurs : <https://jestjs.io/fr/docs/expect>

5. Exécuter le test

```
npm run test
```

```
> testjest@1.0.0 test
> jest --coverage --verbose

PASS ./factorielle.test.js
  ✓ factorielle(2) (2 ms)
  ✓ factorielle(3)
  ✓ factorielle(4) (1 ms)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
factorielle.js	100	100	100	100	

```
Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.303 s, estimated 1 s
Ran all test suites.
```

Dans l'exemple tout le code est couvert par le test. Un répertoire *coverage* est généré. Il contient un rapport au format HTML dans le sous-répertoire *lcov-report*.

All files

100% Statements 4/4

100% Branches 4/4

100% Functions 1/1

100% Lines 4/4

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
factorielle.js	100%	4/4	100%	1/1

Un clic sur le nom du fichier JavaScript affiche le fichier avec les lignes couvertes :

All files factorielle.js

100% Statements 4/4

100% Branches 4/4

100% Functions 1/1

100% Lines 4/4

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1 function factorielle(n) {
2   9x   if ((n == 0) || (n == 1)) {
3   3x     return 1;
4     } else {
5   6x     return n * factorielle(n-1);
6   }
7 }
8 1x module.exports = factorielle;
9
10
```

6. Ecrire un test paramétré

Créer le fichier *factorielle.param.test.js* qui contient un test paramétré pour tester la factorielle de 1 à 6 :

```
const factorielle = require('./factorielle');
test.each([ [1, 1], [2, 2], [3, 6], [4, 24], [5, 120], [6, 720] ])(
  'fact %i = %i', (n, expected) => {
    expect(factorielle(n)).toBe(expected);
  }
);
```

Exécuter :

```
npm run test
```

```
> testjest@1.0.0 test
> jest --coverage --verbose

PASS ./factorielle.param.test.js
  ✓ fact 1 = 1 (1 ms)
  ✓ fact 2 = 2
  ✓ fact 3 = 6
  ✓ fact 4 = 24
  ✓ fact 5 = 120 (1 ms)
  ✓ fact 6 = 720

PASS ./factorielle.test.js
  ✓ factorielle(2) (2 ms)
  ✓ factorielle(3) (1 ms)
  ✓ factorielle(4)
```

7. Créer un module de vérification de chaînes

Créer un module *stringcheck.js* qui comporte les fonctions suivantes :

- *isEmpty* retourne vrai si la chaîne reçue en paramètre est vide, une fois les espaces supprimés.
- *isInt* retourne vrai si la chaîne reçue en paramètre peut être convertie en un nombre entier.
- *isSmallPositiveInt* utilise la méthode *isInt* pour vérifier si la chaîne reçue en paramètre peut être convertie en entier. Elle retourne vrai si la chaîne est un entier positif inférieur à 100.

8. Créer le fichier de test pour le module

Créer un fichier *stringcheck.test.js* :

- Réaliser des positifs et négatifs pour *isEmpty*.
- Regrouper les tests liés à *isEmpty* dans un bloc avec *describe*
<https://jestjs.io/fr/docs/api#describename-fn>
- Exécuter le test
- Remplacer les tests par des tests paramétrés pour *isEmpty*.
- Réaliser des tests paramétrés positifs et négatifs pour *isInt*, en les regroupant avec *describe*.
- Réaliser des tests paramétrés positifs et négatifs pour *isSmallPositiveInt* en les regroupant avec *describe*.

Livrables à déposer sur Ametice en indiquant les noms du binôme

- Code des modules et des tests
- Copie d'écran montrant le résultat des tests et la couverture du code de la section 7