

Tests unitaires : JTest

Ce support vient compléter la démonstration faite en cours

Lien du répo distant de la démonstration :

https://github.com/VDRomainDev/test_unitaire_jest.git

À faire : Cloner le projet pour le tester chez vous

Vous aurez besoin d'un gestionnaire de package afin d'installer la librairie, pour ma part j'utilise **yarn**

Note : Placez-vous à la racine du projet pour exécuter la commande :

```
yarn add --dev jest
```

ou vous pouvez utiliser le gestionnaire de package **npm**

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

À lire : Liste des commandes de comparaison :

<https://classic.yarnpkg.com/en/docs/migrating-from-npm#toc-cli-commands-comparison>

Le principe

Un **test unitaire** permet de vérifier le bon fonctionnement d'une petite partie bien précise du code. Il teste une seule chose à la fois.

Pour tester un projet, il va falloir écrire plusieurs "petits" tests unitaires.

Prenons un exemple simple :

1. Une fonction à tester

D'une part, une fonction est utilisée dans un projet, par exemple, voici la fonction `sum()` qui donne la somme de deux nombres donnés en paramètre :

Dans un fichier ***fun.js***

```
function sum(a, b) {
```

```
    return a + b;
  }

  export default sum
```

2. Un test correspondant

D'autre part, un test permet de tester cette fonction `sum()`

Dans un fichier ***fun.test.js***

```
import sum from './fun'

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Un test pour une fonction c'est :

- Une ou des **entrée(s)**
- Un **résultat attendu**
- Un **résultat obtenu**

Si le résultat obtenu est égal au résultat attendu, alors le test est réussi. Sinon le test a échoué.

Pour le test `expect(sum(1, 2)).toBe(3)`

- Une ou des entrée(s) : 1 et 2
- Un résultat attendu : Ici le résultat attendu est 3
- Un résultat obtenu : c'est le retour de la fonction `sum(1,2)`

3. Lancer les tests

Pour préciser quels sont les tests à effectuer, il faut le préciser dans le "package.json", puis lancer la commande `yarn test`

Dans un fichier `package.json`

```
"scripts": {
  "test": "jest"
}
```

Au lancement des tests, le logiciel vous indique les tests qui ont réussi et ce qui ont échoués. Ce qui permet de trouver les erreurs et les corriger plus rapidement.

4. Interprétation des résultats

```
$ jest
PASS assets/js/helpers/fun.test.js
  ✓ adds 1 + 2 to equal 3 (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.935 s
Ran all test suites.
Done in 2.36s.
```

Normalement le test devrait réussir. (Ci erreur, voir annexe à la dernière page)

À lire : La documentation officielle : <https://jestjs.io/fr/docs/getting-started>

Ils existent une large panoplie de fonction pour écrire des tests, pour commencer concentrer vous sur les assertions de base, qui permettent de tester :

- l'égalité
- la nullité
- l'ordre
- ou la définition

du résultat attendu.

Une **assertion** est une phrase affirmative ou négative que l'on soutient comme vraie.

Exemple :

"La somme de 1 et 2 est égale à 3" est une assertion qui a pour valeur VRAI.

Attention :

"La somme de 1 et 2 n'est pas égale à 0" est une assertion qui a pour valeur VRAI, même si la phrase est négative.

Si vous voulez voir ce qu'il se passe si le test échoue, vous devez écrire une assertion qui est fausse.

"La somme de 1 et 2 est égale à 0" est une assertion qui a pour valeur FAUX.

À lire : <https://jestjs.io/fr/docs/using-matchers>

Exercices

Exercice 1 : SLUG

1. Créer une fonction qui permet de convertir une phrase quelconque en “slug WordPress”, pour cela les espaces doivent être retirés et les lettres doivent être en minuscule.
2. Écrire des testes unitaires qui permettent de tester les assertions suivantes.

Entrée	Résultat attendu	Résultat obtenu
Le petit chat gris	le-petit-chat-gris	
LE GROS CHIEN JAUNE	le-gros-chien-jaune	
Le rouge-gorge Chantant	le-rouge-gorge-chantant	

Exercice 2 : EMAIL

1. Créer une fonction qui permet de vérifier si un email est valide ou non. Cette fonction devra retourner un booléen.
2. Écrire des testes unitaires qui permettent de tester les assertions suivantes.

Entrée	Résultat attendu	Résultat obtenu
toto@gmail.com	true	
toto@gmail.fr	true	
toto@gmail	false	
toto.gmail.com	false	

3. Le plus difficile dans les tests unitaires, c'est de les conceptualiser. Entraînez-vous en imaginant 10 autres assertions pour cette fonction.

Exercice 3 : Date

Si vous avez compris le principe, vous pouvez écrire vous-même des tests unitaires. Avec l'expérience, on se rend compte qu'il est même plus facile d'écrire les tests avant de coder la fonction. Le but de cet exercice est de coder une fonction qui est capable de vérifier la validité d'une date.

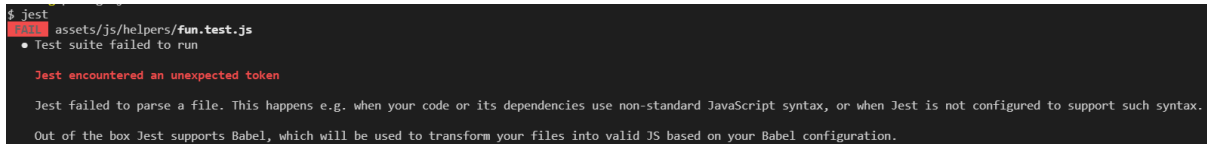
Travail à faire :

1. Trouver au moins une dizaine d'assertions à tester. Exemple : "99/99/9999 n'est pas une date valide"
2. Écrire des tests unitaires.
3. Coder la fonction en lançant les tests au fur et à mesure de son développement, jusqu'à obtenir 100% de réussite.

Bonus : Vérifier aussi les années bissextiles.

Annexe

Si vous avez cette erreur-là :



```
$ jest
FAIL assets/js/helpers/fun.test.js
  ● Test suite failed to run

    Jest encountered an unexpected token

    Jest failed to parse a file. This happens e.g. when your code or its dependencies use non-standard JavaScript syntax, or when Jest is not configured to support such syntax.

    Out of the box Jest supports Babel, which will be used to transform your files into valid JS based on your Babel configuration.
```

Ceci est dû, car Jest ne peut pas comprendre le système d'import de module des normes ES6

Pour aider Jest, nous allons utiliser le compiler Babel avec yarn (toujours dans le terminal)

<https://jestjs.io/docs/getting-started#using-babel>

```
yarn add --dev babel-jest @babel/core @babel/preset-env
```

Une fois les dépendances installer, votre fichier package.json devrait ressembler à ceci :



```
{
  "devDependencies": {
    "@babel/core": "^7.17.8",
    "@babel/preset-env": "^7.16.11",
    "babel-jest": "^27.5.1",
    "jest": "^27.5.1"
  }
}
```

```
},  
  "scripts": {  
    "test": "jest"  
  }  
}
```

Nous allons créer aussi un fichier à la racine du projet afin d'aider babel à se mettre sur votre version de NodeJs

// babel.config.js

Et dedans nous allons y coller ce code suivant :

```
module.exports = {  
  presets: [['@babel/preset-env', {targets: {node: 'current'}}]],  
};
```

Nous pouvons relancer un “yarn test” et cette fois-ci, “jest” devrait pouvoir résoudre les problèmes d’import et nous avons effectué les tests demandés.

Liens utiles

Tutoriel en video :

<https://grafikart.fr/tutoriels/jest-test-framework-1202>

Pour tester les expressions régulières :

<https://regex101.com/>