Cours : Automatiser les tests front-end avec Jest (JS classique)

Introduction : Les tests unitaires et l'automatisation

Les **tests unitaires** sont des tests qui vérifient le comportement d'une petite unité de code, souvent une fonction ou un module, de manière isolée. Ils permettent de s'assurer que chaque partie du code fonctionne comme prévu.

L'**automatisation des tests** consiste à écrire des scripts qui exécutent automatiquement ces tests, sans intervention manuelle. Cela permet de gagner du temps, de détecter rapidement les régressions et d'améliorer la qualité du code.

Pourquoi utiliser Jest ?

- Jest est un framework de test JavaScript populaire, maintenu par Facebook.
- Il est simple à installer et à configurer, même pour du JavaScript "vanilla" (sans framework).
- Jest intègre un simulateur de navigateur via `jsdom` pour tester le DOM.
- Il fournit une syntaxe claire et expressive pour écrire les tests (`test()`, `expect()`, etc.).
- Jest supporte le mocking, les tests asynchrones, les snapshots, et plus encore.
- Grâce à Jest, on peut automatiser efficacement les tests front-end et back-end.

1. Installer Jest

1. Installer Jest en dépendance de développement :

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

2. Installer aussi l'environnement jsdom (si on souhaite test du DOM) :

```
npm install --save-dev jest-environment-jsdom
```

3. Ajouter un script dans package.json pour lancer les tests :

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

2. Configurer Jest pour utiliser jsdom (si besoin)

Créer ou modifier jest.config.js pour préciser l'environnement de test :

```
module.exports = {
  testEnvironment: 'jsdom',
};
```

3. Écrire un test simple

3.1 Notre premier test

Prenom le fichier que l'on veut tester : **math.js**

```
function add(a, b) {
  return a + b;
}

module.exports = add;
```

écrivons maintenant un fichier de test **math.test.js**

```
const add = require('./math');

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

3.2 Lancer les tests

```
npm test
```

Les tests seront effectué et les résultats affichés

3.2.1 Comment Jest détecte les fichiers de test

Jest utilise plusieurs conventions pour reconnaître quels fichiers sont des tests :

- Par défaut, Jest cherche dans le dossier `__tests__/` tous les fichiers avec extension `.js`, `.jsx`, `.ts` ou `.tsx`.
- Il détecte aussi les fichiers qui ont l'extension ou suffixe `.test.js`, `.spec.js`, `.test.ts`, etc.
- Exemple de noms reconnus automatiquement :
    - `math.test.js`

- `user.spec.js`
- `__tests__/app.js`

Tu peux configurer ou changer cette règle dans le fichier `jest.config.js` avec la propriété `testMatch`. Par exemple :

```
module.exports = {
  testMatch: ['**/?(*.)+(spec|test).[jt]s?(x)'],
};
```

3.2.2 Affichage des résultats des tests

Quand tu lances Jest (ex : `npm test`), il affiche dans la console :

- Le nombre total de tests exécutés
- Le nombre de tests réussis, échoués, ou ignorés (skipped)
- Le détail des tests échoués avec le message d'erreur et la pile d'appels
- La durée totale du run des tests

Exemple de sortie simplifiée :

```
 PASS  ./math.test.js
  ✓ adds 1 + 2 to equal 3 (5 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.234 s
Ran all test suites.
```

Options utiles pour l'affichage

- `--watch` : rerun automatiquement les tests à chaque modification des fichiers
- `--verbose` : affiche plus de détails sur chaque test
- `--coverage` : affiche le rapport de couverture de code

Avec ces conventions et options, Jest facilite grandement la gestion et la lecture des tests automatisés.

4. Tester la manipulation du DOM avec Jest

4.1 Test basique

Contenu du fichier à tester **dom.js** :

```
function addParagraph(text) {
  const p = document.createElement('p');
  p.textContent = text;
  document.body.appendChild(p);
}

module.exports = addParagraph;
```

Contenu du fichier de test **dom.test.js** :

```
const addParagraph = require('./dom');

test('adds a paragraph to the document body', () => {
  // Nettoyer le DOM avant test
  document.body.innerHTML = '';

  addParagraph('Hello world');

  const p = document.querySelector('p');
  expect(p).not.toBeNull();
  expect(p.textContent).toBe('Hello world');
  expect(document.body.contains(p)).toBe(true);
});
```

4.2 Charger un fichier HTML dans les tests

On peut lire un fichier HTML et l'injecter dans le DOM simulé par `jsdom` :

```
const fs = require('fs');
const path = require('path');

beforeEach(() => {
  const html = fs.readFileSync(path.resolve(__dirname, 'index.html'), 'utf8');
  document.documentElement.innerHTML = html;
});
```

4.3 Exemple complet

Contenu du fichier **bouton.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test DOM</title>
</head>
<body>
    <button id="btn">Clique-moi</button>
    <p id="text"></p>
</body>
</html>
```

contenu du fichier à tester **bouton.js**

```
document.getElementById('btn').addEventListener('click', () => {
  document.getElementById('text').textContent = 'Bouton cliqué !';
});
```

Contenu du fichier de test **script.test.js**

```
const fs = require('fs');
const path = require('path');

beforeEach(() => {
  const html = fs.readFileSync(path.resolve(__dirname, 'index.html'), 'utf8');
  document.documentElement.innerHTML = html;
  require('./script.js'); // Charger le script pour attacher l'événement
});

test('le texte change quand on clique sur le bouton', () => {
  const button = document.getElementById('btn');
  button.click();

  const text = document.getElementById('text');
  expect(text.textContent).toBe('Bouton cliqué !');
});
```

---

5. Notes importantes

- Jest utilise `jsdom` pour simuler un navigateur.
- Le DOM est remis à zéro avant chaque test via `beforeEach()`.
- Tu peux simuler des événements DOM (`click`, `input`, etc.).
- Le fichier HTML est chargé dans le test pour simuler la structure du document.
- Le script JS est chargé dans le test pour que les gestionnaires d'événements soient attachés.

---

6. Erreur fréquente et solution

Erreur :

Si vos fichiers utilisent des modules (export import), cela peut poser problème.

Cause :

Par défaut Jest utilise les modules de CommonJS utilisé par nodeJS et non pas les modules de base de JS (ES6 module)

Solution :

Modifier les fichiers de paramètrage de Jest et de npm (package.json) pour intégrer les modules ES6. (peut être complexe)

---

7. Ressources

- Jest Documentation
- Jest Environment jsdom
- Node.js fs module
- jsdom GitHub

> Avec ces bases, tu peux automatiser des tests front-end robustes, même sans framework, en simulant le DOM dans Node.js avec Jest.