Les tests unitaires en JavaScript avec mocha

Achref El Mouelhi

Effectuer des tests unitaires avec `mocha`

Comment faire ?

Le rôle des tests unitaires est de s'assurer que les différents fragments de notre projet fonctionnent correctement et donnent les résultats attendus.

Effectuer des tests unitaires avec `mocha`

Comment faire ?

Le rôle des tests unitaires est de s'assurer que les différents fragments de notre projet fonctionnent correctement et donnent les résultats attendus.

Plusieurs frameworks pour JavaScript

- Jasmine
- mocha
- QUnit
- Sinon.JS

mocha

Étapes

- Créer un répertoire projet et se positionner dedans
- Télécharger et installer nodeJS
  `https://nodejs.org/en/download/`
- Utiliser `npm` (node package manager) : le gestionnaire de paquets officiel de `nodeJS`
  - Initialiser npm : exécuter la commande `npm init` dans l'invite de commandes et garder les valeurs par défaut (le `package.json` a été créé dans votre répertoire)
  - Installer mocha avec la commande `npm i mocha --save-dev` ou `npm install -g mocha` (un répertoire `node_modules` a été aussi créé dans votre répertoire si on n'a pas choisi l'installation globale)

mocha

Vérifions l'installation

- Créer un répertoire testFiles dans le répertoire projet (le répertoire qui contiendra tous les fichiers de test)
- Créer un fichier de test `fileTest.js`
- Copier le code suivant dans `fileTest.js`

```
describe('premier test', function(){
    it('affiche quelque chose', function(){
    });
});
```

- Exécuter la commande `mocha`

mocha

Remarques

- On peut aussi modifier la section `scripts` dans `package.json` et remplacer le code existant par le code suivant

```
"scripts": {
    "test": "mocha"
}
```

- Dorénavant, on peut soit exécuter la commande `mocha` soit `npm test`

Exemple

```
// pour importer la librairie assert
var assert = require('assert');
describe('premier test', function(){
    it('should do something', function(){
        var x = 4;
        // verifier si l'assertion x + 2 = 6 est
         vraie
        assert.equal(x + 2, 6);
    });
});
```

Exécuter la commande `mocha`

Exemple

```
var assert = require('assert');
describe('premier test', function(){
    it('should do something', function(){
        var x = 4;
        assert.equal(x + 2, 6);
        assert.equal(x + 3, 6);
    });
});
```

Exécuter la commande `mocha`

Exemple

```
var assert = require('assert');
describe('premier test', function(){
    it('should do something', function(){
        var x = 4;
        assert.equal(x + 2, 6, "le resultat est vrai
         ");
        assert.equal(x + 3, 6, "le resultat est faux
         ");
    });
});
```

Exécuter la commande `mocha`

mocha

Remarque

- Les tests unitaires ont été introduits pour tester et valider nos scripts
- Ce n'est pas pour tester la validité de nos données (données en dur)

Exemple

Créons un fichier `monScript` dans le répertoire `lib` situé dans la racine de mon projet

```
module.exports = {
        moyenne : function(note1, note2){
                return (note1 + note2)/2;
        },
        division : function(note1, note2){
                return note1 / note2;
        }
}
```

Exemple

Testons ces deux fonctions dans le fichier `fileTest.js`

```
var assert = require('assert');
var monScript = require('../lib/monScript');
describe('monScript', function(){
  describe('moyenne', function(){
    it('should␣compute␣a␣correct␣average', function(){
      assert.equal(monScript.moyenne(10,16), 13, "le␣resultat␣
       est␣correct");
      assert.equal(monScript.moyenne(8,13), 10.5, "le␣resultat␣
       est␣correct");
    });
  });
  describe('division', function(){
    it('should␣compute␣a␣correct␣division', function(){
      assert.equal(monScript.division(10,5), 2, "le␣resultat␣
       est␣correct");
      assert.equal(monScript.division(10,3), 3.3, "le␣resultat␣
       est␣correct");
    });
  });
```

Exemple

Corrigeons et améliorons nos codes

```
module.exports = {
        moyenne : function(note1, note2){
                return (note1 + note2)/2;
        },
        division : function(note1, note2){
                if (0 == note2)
                        return Infinity;
                return this.arrondi(note1 / note2);
        },
        arrondi : function(x){
                return Math.round(x);
        }
}
```

Exemple

Re-testons ces deux fonctions dans le fichier `fileTest.js`

```
var assert = require('assert');
var monScript = require('../lib/monScript');
describe('monScript', function(){
  describe('moyenne', function(){
    it('should␣compute␣a␣correct␣average', function(){
      assert.equal(monScript.moyenne(10,16), 13, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.moyenne(8,13), 10.5, "le␣resultat␣est␣
       correct");
    });
  });
  describe('division', function(){
    it('should␣compute␣a␣correct␣division', function(){
      assert.equal(monScript.division(10,5), 2, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.division(10,3), 3, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.division(10,0), Infinity, "le␣resultat␣est
       ␣correct");
    });
  });
});
```

mocha

Remarque

Vous pouvez modifier le mode d'affichage en faisant

- `mocha -R landing`
- `mocha -R dot`
- `mocha -R list`
- `mocha -b` (bail) : il s'arrête dès qu'il détecte une erreur
- `mocha -g moyenne` : moyenne étant le nom qu'on a défini dans `describe` : lancer le test que pour le module `moyenne`
- `mocha --help` : pour explorer la liste des commandes

Exemple

```
var assert = require('assert');
var monScript = require('../lib/monScript');
describe('monScript', function(){
  describe('moyenne', function(){
    it.only('should␣compute␣a␣correct␣average', function(){
      assert.equal(monScript.moyenne(10,16), 13, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.moyenne(8,13), 10.5, "le␣resultat␣est␣
       correct");
    });
  });
  describe('division', function(){
    it('should␣compute␣a␣correct␣division', function(){
      assert.equal(monScript.division(10,5), 2, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.division(10,3), 3, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.division(10,0), Infinity, "le␣resultat␣est
       ␣correct");
    });
  });
});
```

`only` permet d'exécuter seulement ce module

Exemple

```
var assert = require('assert');
var monScript = require('../lib/monScript');
describe('monScript', function(){
  describe('moyenne', function(){
    it.skip('should␣compute␣a␣correct␣average', function(){
      assert.equal(monScript.moyenne(10,16), 13, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.moyenne(8,13), 10.5, "le␣resultat␣est␣
       correct");
    });
  });
  describe('division', function(){
    it('should␣compute␣a␣correct␣division', function(){
      assert.equal(monScript.division(10,5), 2, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.division(10,3), 3, "le␣resultat␣est␣
       correct");
      assert.equal(monScript.division(10,0), Infinity, "le␣resultat␣est
       ␣correct");
    });
  });
});
```

`skip` permet de ne pas effectuer ce test

Exemple

```
var assert = require('assert');
var monScript = require('../lib/monScript');
describe('monScript', function(){
  beforeEach(function() {
    console.log('each');
  });
  describe('moyenne', function(){
    it('should compute a correct average', function(){
      assert.equal(monScript.moyenne(10,16), 13, "resultat correct");
    });
  });
  describe('division', function(){
    it('should compute a correct division', function(){
      assert.equal(monScript.division(10,5), 2, "resultat correct");
    });
  })
});
```

- `beforeEach` se lance avant l'exécution de chaque test
- `before` se lance une fois avant l'exécution des tests
- `afterEach` se lance après l'exécution de chaque test
- `after` se lance une fois après l'exécution des tests

Exemple

On peut aussi tester l'inégalité

```
var assert = require('assert');
var monScript = require('../lib/monScript');
describe('monScript', function(){
  describe('division', function(){
    it('should␣compute␣a␣correct␣division', function
     (){
      assert.notEqual(monScript.division(10,5), 3, "
       le␣resultat␣est␣correct");
      assert.notEqual(monScript.division(10,4), 2, "
       le␣resultat␣est␣correct");
    });
  });
});
```

Exemple

```
var assert = require('assert');
let tab1 = [1, 2, 3, 4, 5], tab2 = [1, 2, 3, 4, 5];
describe('monScript', function(){
    describe('comparaison', function(){
        it('should␣check␣array', function(){
                assert.deepEqual(tab1, tab2, "le␣resultat␣est␣
                 vrai");
                assert.notEqual(tab1, tab2, "le␣resultat␣est␣
                 vrai");
                });
        });
});
```

- `deepEqual` et `notDeepEqual` pour comparer des tableaux, des objets...
- `strictEqual` et `notStrictEqual` pour faire une comparaison stricte (par exemple entre caractère et entier, un booléen et un entier...)

Exemple

```
var assert = require('assert');
var personne1 = { "name":"jack", "age":"21" };
var personne2 = { "name":"jack", "age":"21" };
describe('monScript', function(){
  describe('comparaison', function(){
    it('should check object', function(){
      assert.deepEqual(personne1, personne2, "le resultat est vrai");
      assert.notEqual(personne1, personne2, "le resultat est vrai");
    });
  });
});
```

Exemple

```
var assert = require('assert');
var personne1 = { "name":"jack", "age":"21" };
var personne2 = { "name":"jack", "age":"21" };
describe('monScript', function(){
  describe('comparaison', function(){
    it('should check object', function(){
      assert.deepEqual(personne1, personne2, "le resultat est vrai");
      assert.notEqual(personne1, personne2, "le resultat est vrai");
    });
  });
});
```

```
var assert = require('assert');
describe('monScript', function(){
  describe('comparaison', function(){
    it('should check strict equality', function(){
      assert.notStrictEqual(0, false, 'pas les memes');
      assert.equal(0, false, 'les memes');
      assert.notStrictEqual(1, '1', 'pas les memes');
    });
  });
});
```

mocha

Remarque

- `it` a un alias `specify`
- `describe` a un alias `context`