

Test en node.js

Mocha, Chai, Sinon

César Augusto Rodriguez

Qualité du logiciel et métriques

19 Septembre 2024

Ordre de cette présentation

- 1 Introduction
- 2 Test-Driven Development
- 3 Comment faire des tests ?
- 4 Mocha & Chai
- 5 Sinon
- 6 Conclusion

Introduction

Qu'est-ce que c'est ?

C'est le processus de vérification individuelle des modules et fonctions dans notre code pour s'assurer qu'ils fonctionnent comme prévu.

Pourquoi est-ce important ?

- Garantir le bon fonctionnement du code.
- Prévenir les bugs avant qu'ils ne se produisent.
- La base du Test-Driven Development (TDD).

Test-Driven Development

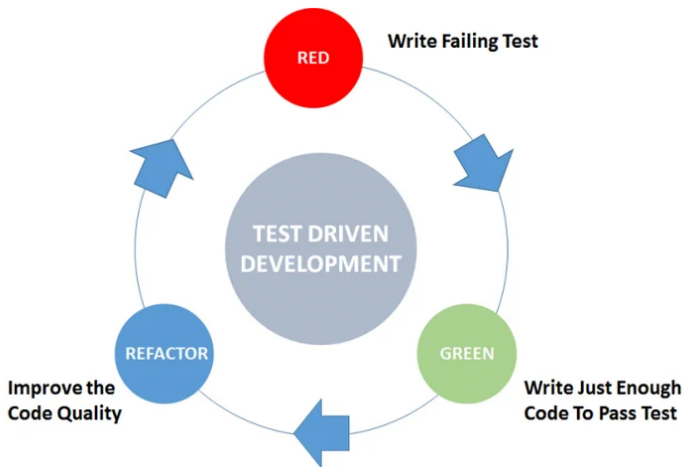


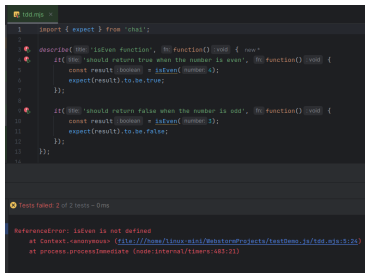
Figure: Étapes du TDD

Example TDD

Imaginons que nous voulons faire un programme simple qui vérifie si un nombre est pair ou non.

Première étape **Rouge**:

- Écrivons les tests d'abord sans même pas penser à notre fonction.
- Les tests vont échouer car notre fonction `isEven()` n'existe pas.



```
1 import { expect } from 'chai';

2 describe('isEven function', () => {
3   it('should return true when the number is even', () => {
4     const result = isEven(4);
5     expect(result).toBe(true);
6   });
7
8   it('should return false when the number is odd', () => {
9     const result = isEven(3);
10    expect(result).toBe(false);
11  });
12 });

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Tests failed: 2 of 2 tests - 0ms

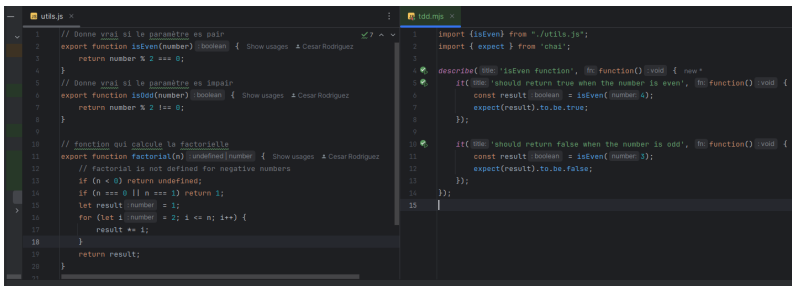
ReferenceError: isEven is not defined
at Context.<anonymous> (file:///home/linux/.vs/DevstoreProjects/testDemo.js/test_mis.3.24)
at process.processImmediate (node:internal/timers:463:21)

Figure: Red step

Example TDD

Deuxième étape Vert:

- On fait les modules nécessaires pour faire passer nos tests, sans aller plus loin. La fonction `isEven()` est faite sur `utils.js`

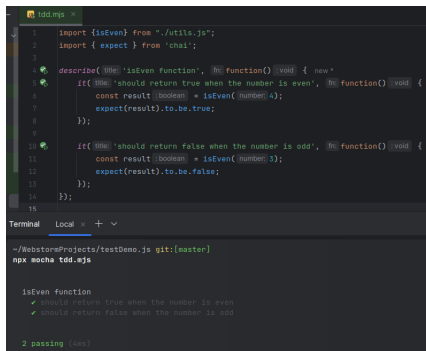


```
utils.js
1 // Donne vrai si le paramètre es pair
2 export function isEven(number) {
3   return number % 2 === 0;
4 }
5 // Donne vrai si le paramètre es impair
6 export function isOdd(number) {
7   return number % 2 !== 0;
8 }
9
10 // fonction qui calcule la factorielle
11 export function factorial(n) {
12   // factorial is not defined for negative numbers
13   if (n < 0) return undefined;
14   if (n === 0 || n === 1) return 1;
15   let result = 1;
16   for (let i = 2; i <= n; i++) {
17     result *= i;
18   }
19   return result;
20 }
21
index.mjs
1 import { isEven } from './utils.js';
2 import { expect } from 'chai';
3
4 describe('isEven function', function() {
5   it('should return true when the number is even', function() {
6     const result = isEven(4);
7     expect(result).to.be.true;
8   });
9
10   it('should return false when the number is odd', function() {
11     const result = isEven(3);
12     expect(result).to.be.false;
13   });
14 });
```

Figure: Green step

- Les tests vont passer maintenant.

Example TDD



```
1 import {isEven} from './utils.js';
2 import {expect} from 'chai';
3
4 describe('isEven function', function() {
5   it('should return true when the number is even', function() {
6     const result = isEven(4);
7     expect(result).toBe(true);
8   });
9
10  it('should return false when the number is odd', function() {
11    const result = isEven(3);
12    expect(result).toBe(false);
13  });
14 });
```

Terminal

```
~/WebstormProjects/testDemo.js git:(master)
npx mocha tdd.mjs

isEven function
  ✓ should return true when the number is even
  ✓ should return false when the number is odd

2 passing (4ms)
```

Troisième étape **Refactoring**:

- On améliore la qualité du code, et on peut refaire le cycle à nouveau.
- Dans ce cas simple on n'a pas besoin.

Comment faire des tests...

Manuellement

- Écrire nos propres fonctions.
- Utilisation personnalisée des fonctions d'affichage, ex. `console.log()`.
- Écrire du code spécifique pour chaque module.
- **Avantages :**
 - Fonctionne bien dans des cas particuliers.
 - Peut être idéal là où les librairies ne sont pas disponibles.
 - Exemple : petits projets, impossibilité d'utiliser une librairie...
 - Nécessite moins de connaissances sur des librairies externes.
- **Inconvénients :**
 - Risque de copier-coller, mauvais patrons de conception.
 - Difficile à gérer avec un grand nombre de modules.

Demo test manuel

```
Project
├── testDemo.js
├── node_modules
├── countLength.js
├── manualTest.js
├── mocha_chai.mjs
├── package.json
├── package-lock.json
├── README.md
├── unitTest.mjs
├── utils.js
├── External Libraries
└── Scratches and Consoles
```

```
// Fonction qui retourne la longueur d'une chaîne
export function countLength(str) {
  let length = 0;

  // boucler jusqu'à la fin de la chaîne
  while (str[length] !== undefined) {
    length++;
  }

  return length; // Return the counted length
}
```

```
import { countLength } from './countLength.js';

// Programme de test
let myString = "Hello, world!"; // chaîne de test
let stringLength = countLength(myString); // appelons countLength

console.log("La longueur devrait être égale à " + myString.length);
console.log("La longueur donnée est " + stringLength);

if (myString.length === stringLength) {
  console.log("Ce test est réussi");
} else {
  console.log("Test échoué");
}
```

```
Run manualTest.js
/snap/bin/node /home/linux-mini/WebstormProjects/testDemo.js/manualTest.js
La longueur devrait être égale à 13
La longueur donnée est 13
Ce test est réussi
Process finished with exit code 0
```

Figure: Exemple d'un test manuel

Tests unitaires avec des librairies

- À l'aide de *frameworks* comme Mocha, Chai, Sinon, etc.
- Plus facile à lire et à entretenir.
- Open-source, maintenues par la communauté.
- **Avantages:**
 - Permet de faire une suite de tests dans un seul bloque.
 - `describe()`, `it()`.
 - Permettent de faire de tests selon les résultats attendus, ex:
 - `assert.equal()`, `assert.isNull()`, etc...
 - Permet de compter le nombre d'appels à une fonction `spy()`.
- **Désavantages**
 - Demande des connaissances sur ces librairies.
 - **Maintenance**
 - Configuration initiale de la librairie,
 - Mise à jour de la librairie, etc

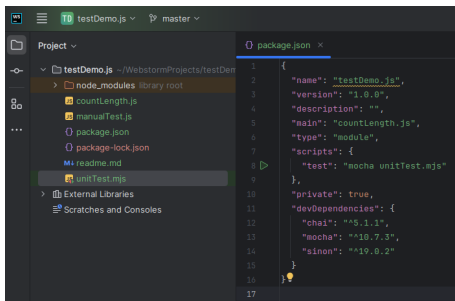
Frameworks in JavaScript

Mocha, Chai et Sinon (configuration)

Installation des bibliothèques

```
npm install mocha chai sinon --save-dev
```

Bien configurer votre fichier json avec "type": "module" et "test": "mocha nom_du_fichier"



The screenshot shows a code editor with a project structure on the left and a `package.json` file open on the right. The project structure includes a `testDemo.js` file and a `node_modules` directory containing `countLength.js`, `manualTest.js`, `package.json`, `package-lock.json`, `readme.md`, and `unitTest.mjs`. The `package.json` file contains the following JSON:

```
{
  "name": "testDemo.js",
  "version": "1.0.0",
  "description": "",
  "main": "countLength.js",
  "type": "module",
  "scripts": {
    "test": "mocha unitTest.mjs"
  },
  "private": true,
  "devDependencies": {
    "chai": "^5.1.1",
    "mocha": "^10.7.3",
    "sinon": "^19.0.2"
  }
}
```

Figure: Exemple de projet JSON

Mocha

- Possiblement la librairie la plus populaire de Node.js pour les test unitaires.
- Permet l'organisation de tests dans de blocs grâce aux fonctions `describe()`, `it()`.
- Souvent accompagné de Chai (prochaines slides).
- [Site officiel Mocha](#).
- Extension `.mjs`

Chai

- Permet l'utilisation des assertions sur les test.
 - Par exemple `assert.equal(factorial(5), 120); // 5! = 120`
- [Site officiel de Chai](#)

Demo Mocha & Chai

```
Project - testDemo.js - Webstorm
  node_modules library
  countLength.js
  manualTest.js
  mocha_chai.mjs
  package.json
  package-lock.json
  readme.md
  unitTest.mjs
  utils.js
External Libraries
Scratches and Consoles

countLength.js
1 // Fonction qui retourne la longueur d'une chaîne
2 export function countLength(str) {
3   let length = 0;
4
5   // bouclier jusqu'à la fin de la chaîne
6   while (str[length] !== undefined) {
7     length++;
8   }
9
10  return length; // Return the counted length
11 }
12

unitTest.mjs
1 import { countLength } from './countLength.js';
2 import { expect } from 'chai'; // Utilisation de import pour chai
3
4 // Début de la suite de tests Mocha
5 describe('Voici des test simples', function() {
6
7   it('devrait retourner la longueur correcte pour une chaîne donnée', function() {
8
9     // Chaîne de test
10    let myString = 'Hello, world!';
11    let stringLength = countLength(myString);
12
13    // Assertion : vérifier si la longueur calculée est égale à la longueur réelle
14    expect(stringLength).to.equal(myString.length);
15  });
16
17   it('Voici un dummy test', function() {
18     let a = 2;
19     let b = 3;
20     expect(a + b).equal(5);
21   });
22
23   // On pourrait continuer à faire de it dans ce block
24 });

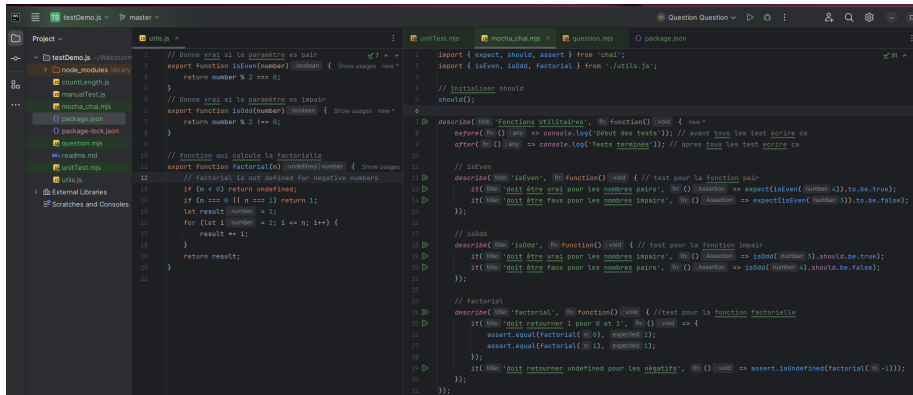
Terminal Local
~/WebstormProjects/testDemo.js git:[master]
npx mocha unitTest.mjs

countLength Function
  ✔ devrait retourner la longueur correcte pour une chaîne donnée
  ✔ voici un dummy test

2 passing (4ms)
```

Figure: Demo simple plusieurs tests describe(), it()

Démo Chai



```
1 // Donne vrai si le paramètre es pair
2 export function isEven(number) {
3   return number % 2 === 0;
4 }
5 // Donne vrai si le paramètre es impair
6 export function isOdd(number) {
7   return number % 2 !== 0;
8 }
9
10 // fonction qui calcule la factorielle
11 export function factorial(n) {
12   // factorial is not defined for negative numbers
13   if (n < 0) return undefined;
14   if (n === 0 || n === 1) return 1;
15   let result = number + 1;
16   for (let i = number + 2; i <= n; i++) {
17     result *= i;
18   }
19   return result;
20 }
21
```

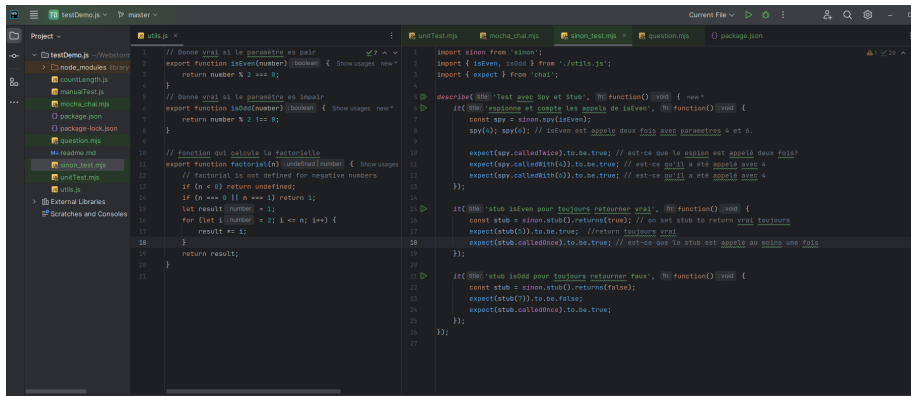
```
1 import { expect, should, assert } from 'chai';
2 import { isEven, isOdd, factorial } from './utils.js';
3
4 // Initialiser should
5 should();
6
7 describe('Fonctions Utilitaires', () => {
8   // before
9   before(() => console.log('Début des tests')); // avant tous les test écrire ça
10  after(() => console.log('Tests terminés')); // après tous les test écrire ça
11
12  // isEven
13  describe('isEven', () => { // test pour la fonction pair
14    it('doit être vrai pour les nombres pairs', () => {
15      expect(isEven(4)).to.be.true;
16    });
17    it('doit être faux pour les nombres impairs', () => {
18      expect(isEven(3)).to.be.false;
19    });
20  });
21
22  // isOdd
23  describe('isOdd', () => { // test pour la fonction impair
24    it('doit être vrai pour les nombres impairs', () => {
25      expect(isOdd(3)).to.be.true;
26    });
27    it('doit être faux pour les nombres pairs', () => {
28      expect(isOdd(4)).to.be.false;
29    });
30  });
31
32  // factorial
33  describe('factorial', () => { // test pour la fonction factorielle
34    it('doit retourner 1 pour 0 et 1', () => {
35      expect(factorial(0)).to.equal(1);
36      expect(factorial(1)).to.equal(1);
37    });
38    it('doit retourner undefined pour les négatifs', () => {
39      expect(isUndefined(factorial(-1))).to.be.true;
40    });
41  });
42
```

Figure: Exemple détaillé - Chai

Librairie intéressant pour simuler le comportement des fonctions.

- `spy()` permet de compter le nombre d'appel sur une fonction, et quel type d'arguments ont été utilisés.
- `stub()` permet de simuler le résultat d'une fonction
 - Par exemple si on doit faire de test sur un module qui utilise une dépendence qu'on ne s'intéresse pas.
 - lorsqu'une fonction veut forcer un erreur sur une fonction por analyser son comportement.
- [Site Web officiel de Sinon](#)

Demo Sinon



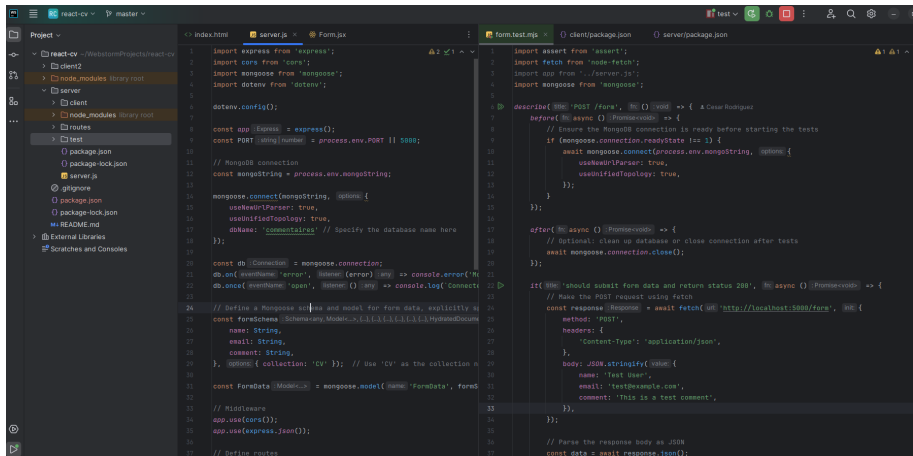
The screenshot shows a code editor with a project structure on the left and two code files open. The project structure includes a 'testDemo.js' file and a 'utils.js' file. The 'utils.js' file contains a function 'isEven' and a 'factorial' function. The 'testDemo.js' file contains a 'describe' block with two 'it' blocks, each with a 'spy' or 'stub' and an 'expect' statement. The 'utils.js' file contains the implementation of the 'isEven' and 'factorial' functions.

```
// utils.js
1 // Donne vrai si le paramètre es pair
2 export function isEven(number) : boolean {
3   return number % 2 === 0;
4 }
5 // Donne vrai si le paramètre es impair
6 export function isOdd(number) : boolean {
7   return number % 2 !== 0;
8 }
9 // fonction qui calcule la factorielle
10 export function factorial(n : undefined | number) {
11   // factorial is not defined for negative numbers
12   if (n < 0) return undefined;
13   if (n === 0 || n === 1) return 1;
14   let result : number = 1;
15   for (let i : number = 2; i <= n; i++) {
16     result *= i;
17   }
18   return result;
19 }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
// testDemo.js
1 import sinon from 'sinon';
2 import { isEven, isOdd } from './utils.js';
3 import { expect } from 'chai';
4
5 describe('Test avec Spy et Stub', () => {
6   it('l\'espionne et compte les appels de isEven', () => {
7     const spy = sinon.spy(isEven);
8     spy(4); spy(6); // isEven est appelé deux fois avec paramètres 4 et 6.
9
10    expect(spy.calledTwice).to.be.true; // est-ce que le espion est appelé deux fois?
11    expect(spy.calledWith(4)).to.be.true; // est-ce qu'il a été appelé avec 4
12    expect(spy.calledWith(6)).to.be.true; // est-ce qu'il a été appelé avec 6
13  });
14
15   it('l\'stub isEven pour toujours retourner vrai', () => {
16     const stub = sinon.stub().returns(true); // on set stub to return vrai toujours
17     expect(stub(3)).to.be.true; //return toujours vrai
18     expect(stub.calledOnce).to.be.true; // est-ce que le stub est appelé au moins une fois
19   });
20
21   it('l\'stub isOdd pour toujours retourner faux', () => {
22     const stub = sinon.stub().returns(false);
23     expect(stub(7)).to.be.false;
24     expect(stub.calledOnce).to.be.true;
25   });
26 });
```

Figure: Exemple Sinon

Projet personnel Demo



```
Project >
├── react-cv - /Users/rodriguez/Projects/react-cv
├── client2
├── node_modules library root
├── server
│   ├── client
│   ├── routes
│   ├── node_modules library root
│   ├── package.json
│   ├── package-lock.json
│   ├── server.js
│   ├── .gitignore
│   ├── package.json
│   ├── package-lock.json
│   ├── README.md
│   └── External Libraries
└── Scratches and Consoles
```

```
index.html
1 import express from 'express';
2 import cors from 'cors';
3 import mongoose from 'mongoose';
4 import dotenv from 'dotenv';
5
6 dotenv.config();
7
8 const app = express();
9 const PORT = (String)process.env.PORT || 5000;
10
11 // MongoDB connection
12 const mongoString = process.env.mongoString;
13
14 mongoose.connect(mongoString, {
15   useNewUrlParser: true,
16   useUnifiedTopology: true,
17   dbName: 'commentaires' // Specify the database name here
18 });
19
20 const db = mongoose.connection;
21 db.on('error', (error) => console.error('MongoDB connection error: ', error));
22 db.once('open', () => console.log('Connected to MongoDB'));
23
24 // Define a Mongoose schema and model for form data, explicitly as
25 const formSchema = Schema({
26   name: String,
27   email: String,
28   comment: String,
29 }, {
30   collection: 'CV'
31 }); // Use 'CV' as the collection name
32
33 const FormData = mongoose.model('FormData', formSchema);
34
35 // Middleware
36 app.use(cors());
37 app.use(express.json());
38
39 // Define routes
```

```
form.test.mjs
1 import assert from 'assert';
2 import fetch from 'node-fetch';
3 import app from './server.js';
4 import mongoose from 'mongoose';
5
6 describe('POST /form', () => {
7   // Ensure the MongoDB connection is ready before starting the tests
8   before(async () => {
9     if (mongoose.connection.readyState !== 1) {
10       await mongoose.connect(process.env.mongoString, {
11         useNewUrlParser: true,
12         useUnifiedTopology: true,
13       });
14     }
15   });
16
17   after(async () => {
18     // Optional: clean up database or close connection after tests
19     await mongoose.connection.close();
20   });
21
22   it('should submit form data and return status 200', async () => {
23     // Make the POST request using fetch
24     const response = await fetch('http://localhost:5000/form', {
25       method: 'POST',
26       headers: {
27         'Content-Type': 'application/json',
28       },
29       body: JSON.stringify({
30         name: 'Test User',
31         email: 'test@example.com',
32         comment: 'This is a test comment',
33       }),
34     });
35
36     // Parse the response body as JSON
37     const data = await response.json();
38   });
39 });
```

Figure: Projet personnel

Conclusion

- Les test unitaires sont la base de la conception TDD.
- Les test unitaires servent à vérifier la bonne qualité du code et à prévenir des bugs.
- Dans javascript, Mocha, Chai et Sinon sont des librairies assez bonnes pour une telle conception.

Bibliographie

- [Udemy Node.js](#)
- [Développement TDD](#)
- [Chai](#)
- [Mocha](#)
- [Sinon](#)