Test en node.js

Mocha, Chai, Sinon

César Augusto Rodriguez

Qualité du logiciel et métriques

19 Septembre 2024

Université de Montréal

Ordre de cette présentation

Université de Montréal

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).

Université de Montréal

Test-Driven Development

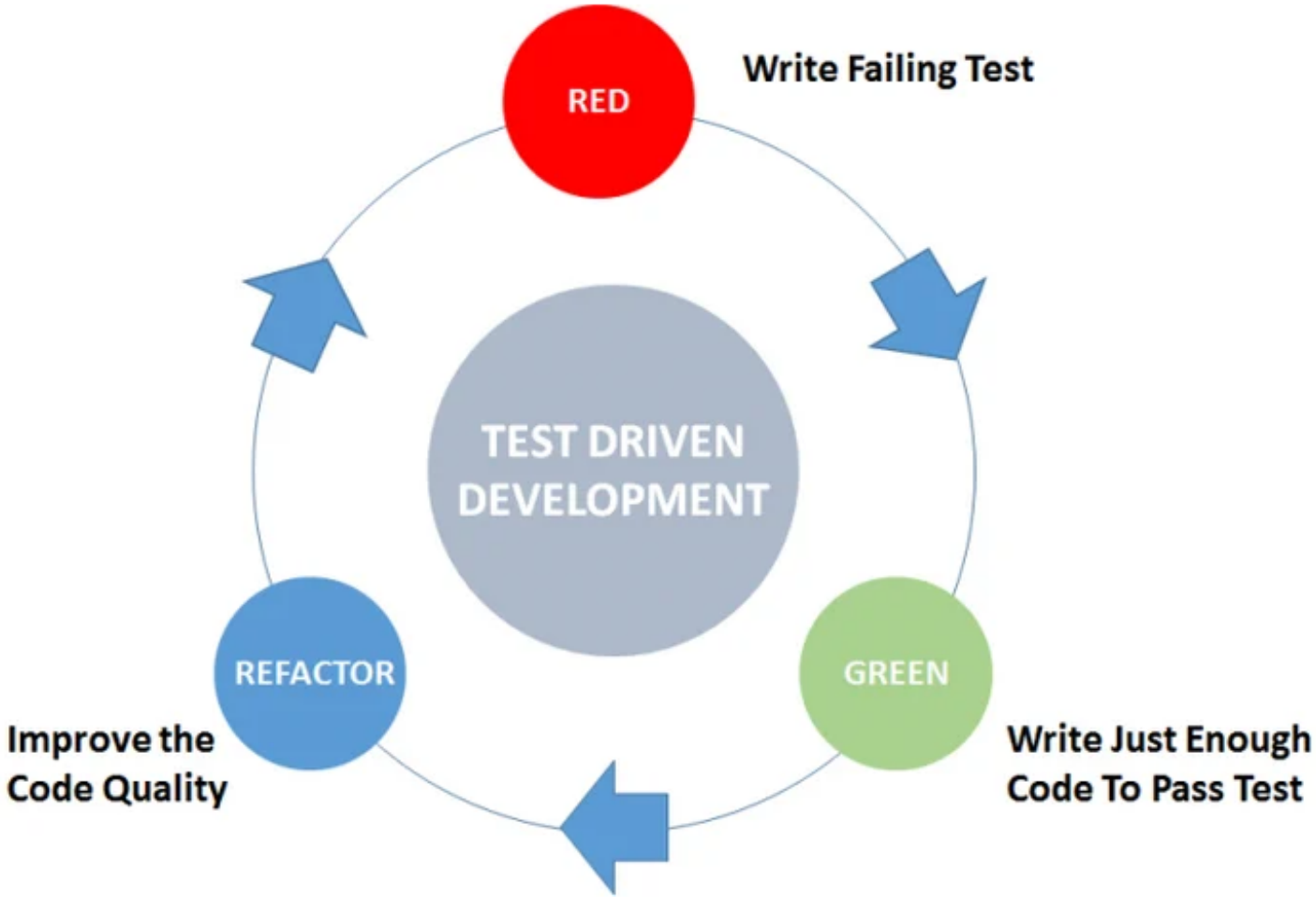


Figure: Étapes du TDD

Université de Montréal

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.

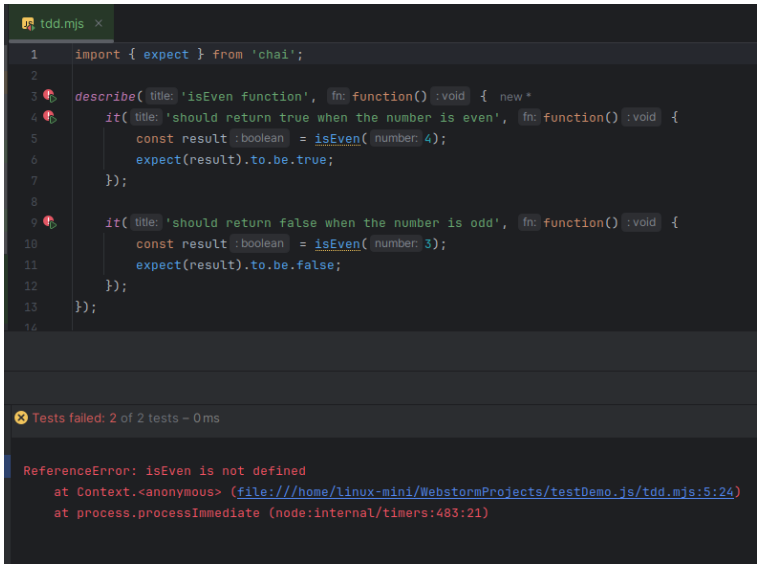


Figure: Red step

Université de Montréal

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`

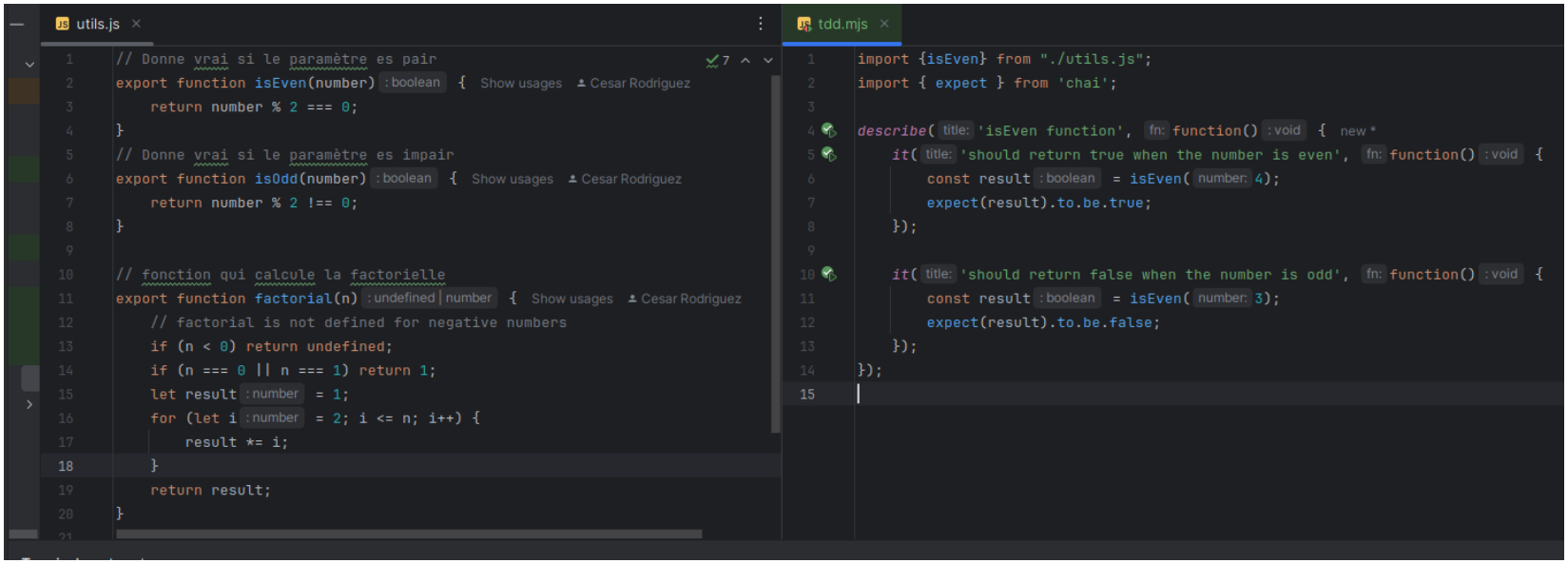
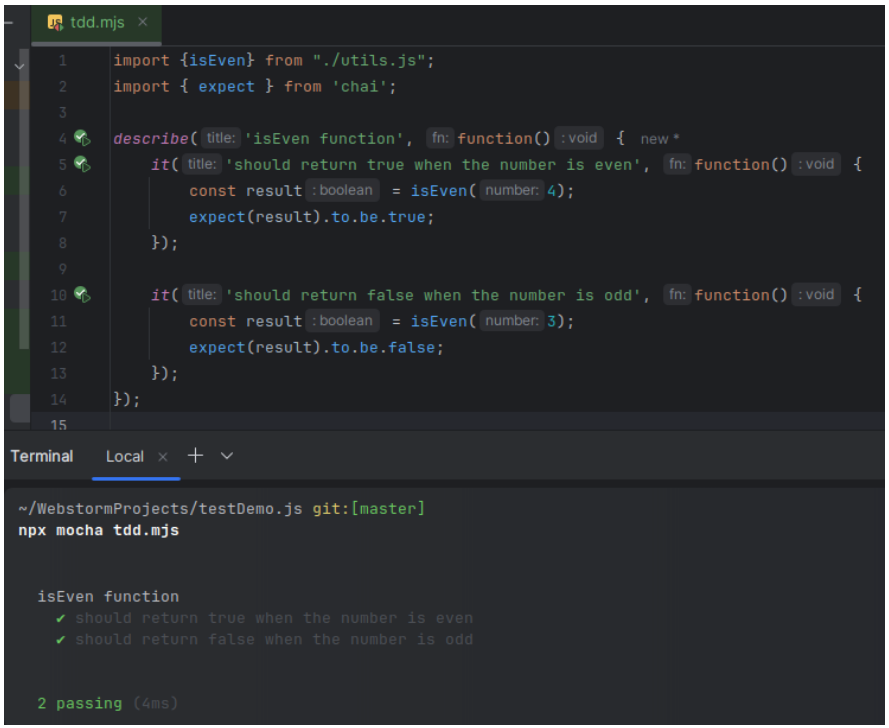


Figure: Green step

- Les tests vont passer maintenant.

Université de Montréal

Example TDD



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.

Université de Montréal

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.

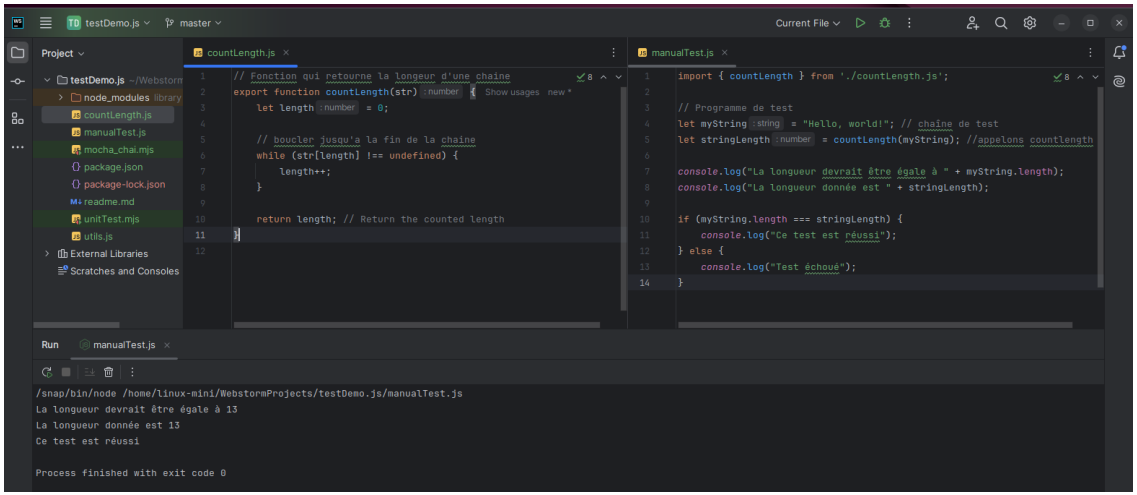


Figure: Exemple d'un test manuel

Université de Montréal

Demo test manuel

- **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.

Université de Montréal

Type de tests

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.

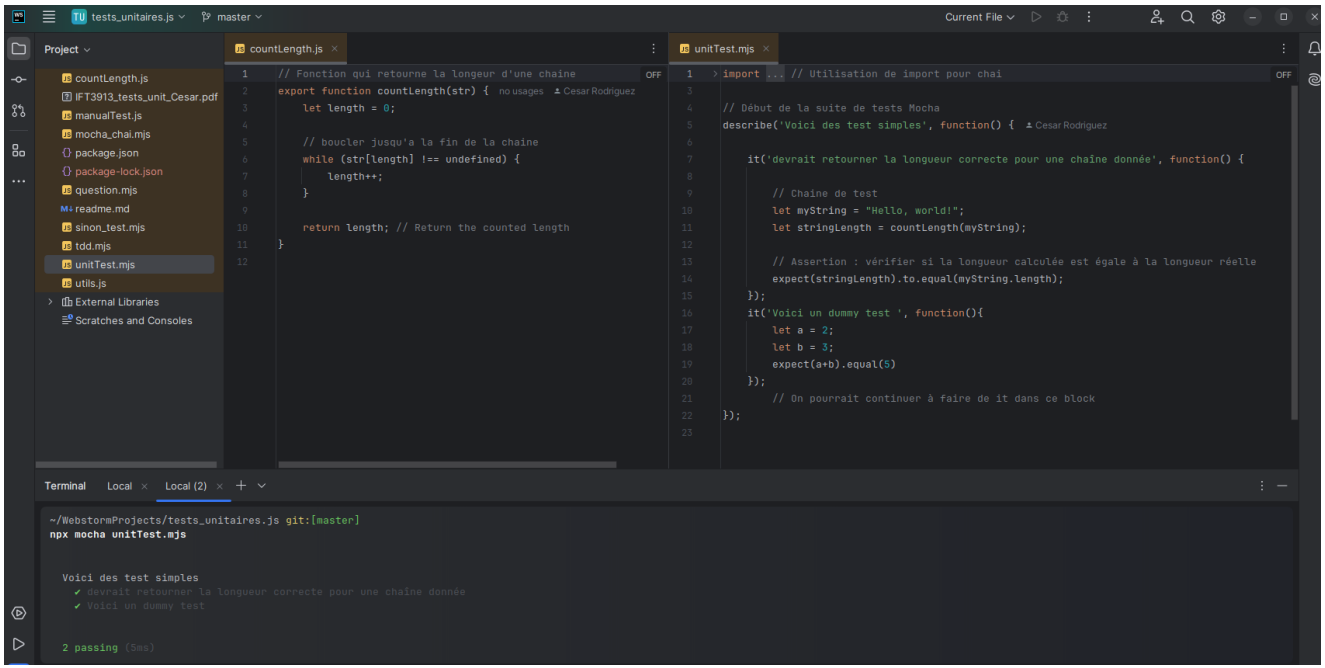


Figure: Demo simple plusieurs tests `describe()`, `it()`

Université de Montréal

Mocha, Chai et Sinon (configuration)

Installation des librairies

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

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

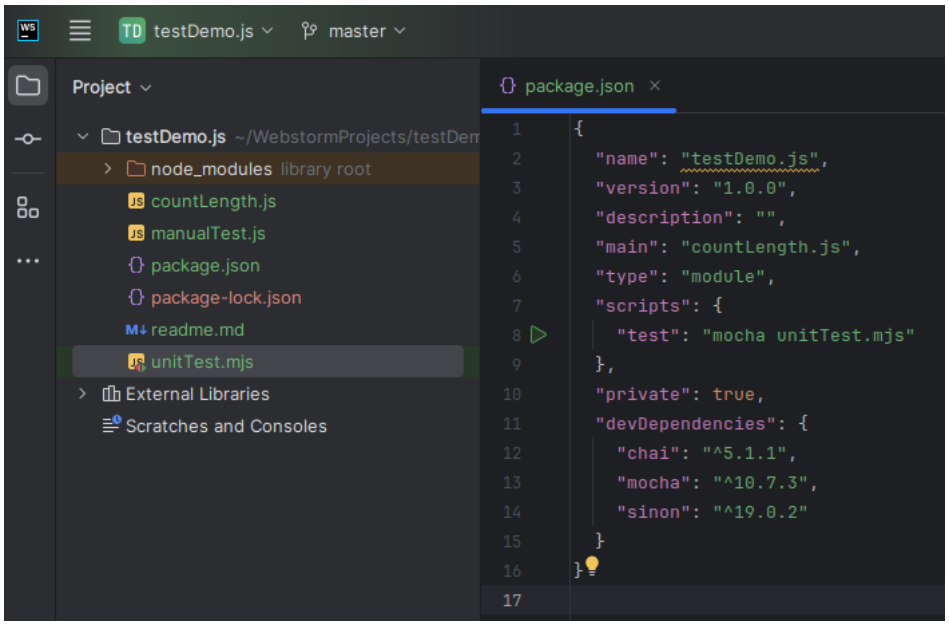


Figure: Exemple de projet JSON

Université de Montréal

Mocha & Chai

Mocha

- Possiblement la librairie la plus populaire de Node.js pour les test unitaires.
- Permet l'organisation de tests dans de bloques 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

Université de Montréal

Avantages vs inconvenients

- **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 libraire,
    - Mise à jour de la librairie, etc

Université de Montréal

Démo Chai

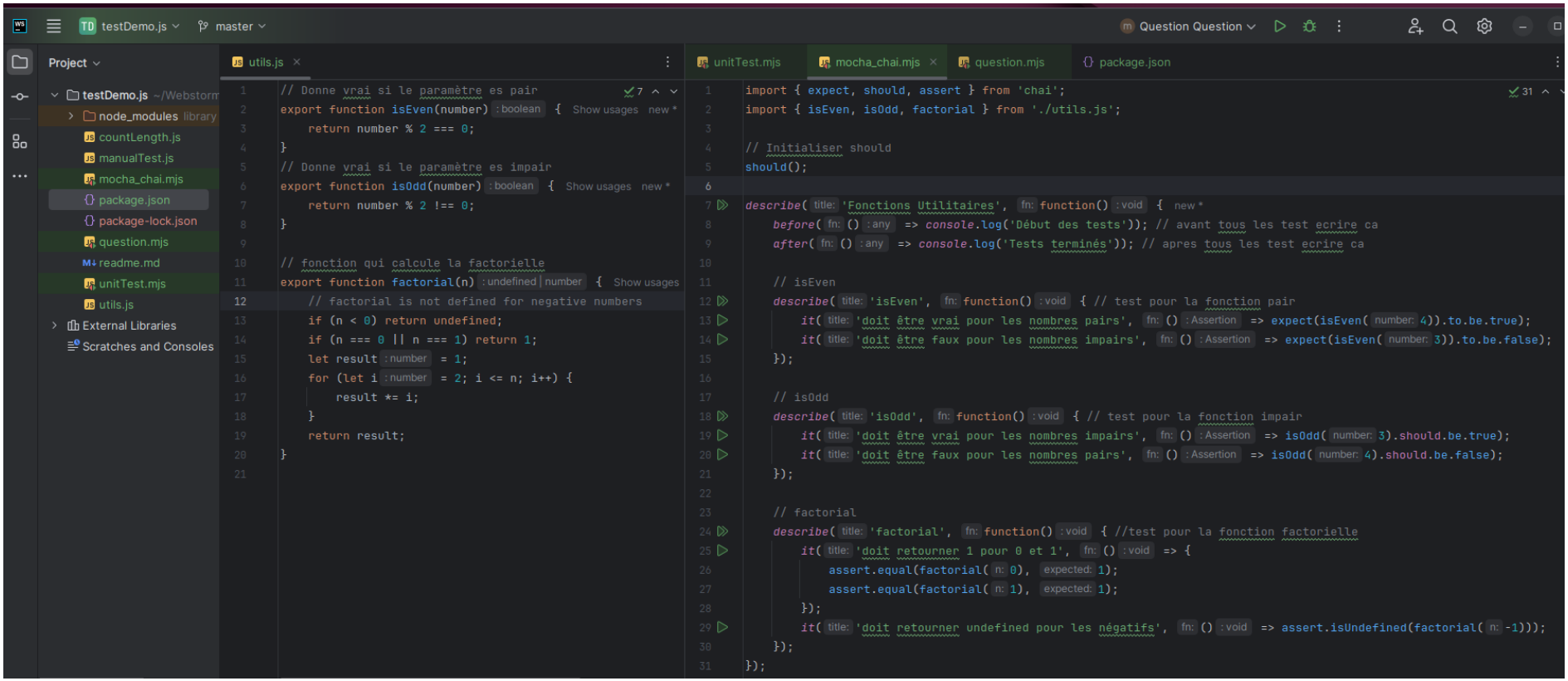


Figure: Exemple détaillé - Chai

Université de Montréal

Sinon

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 dependence 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

Université de Montréal

Demo Sinon

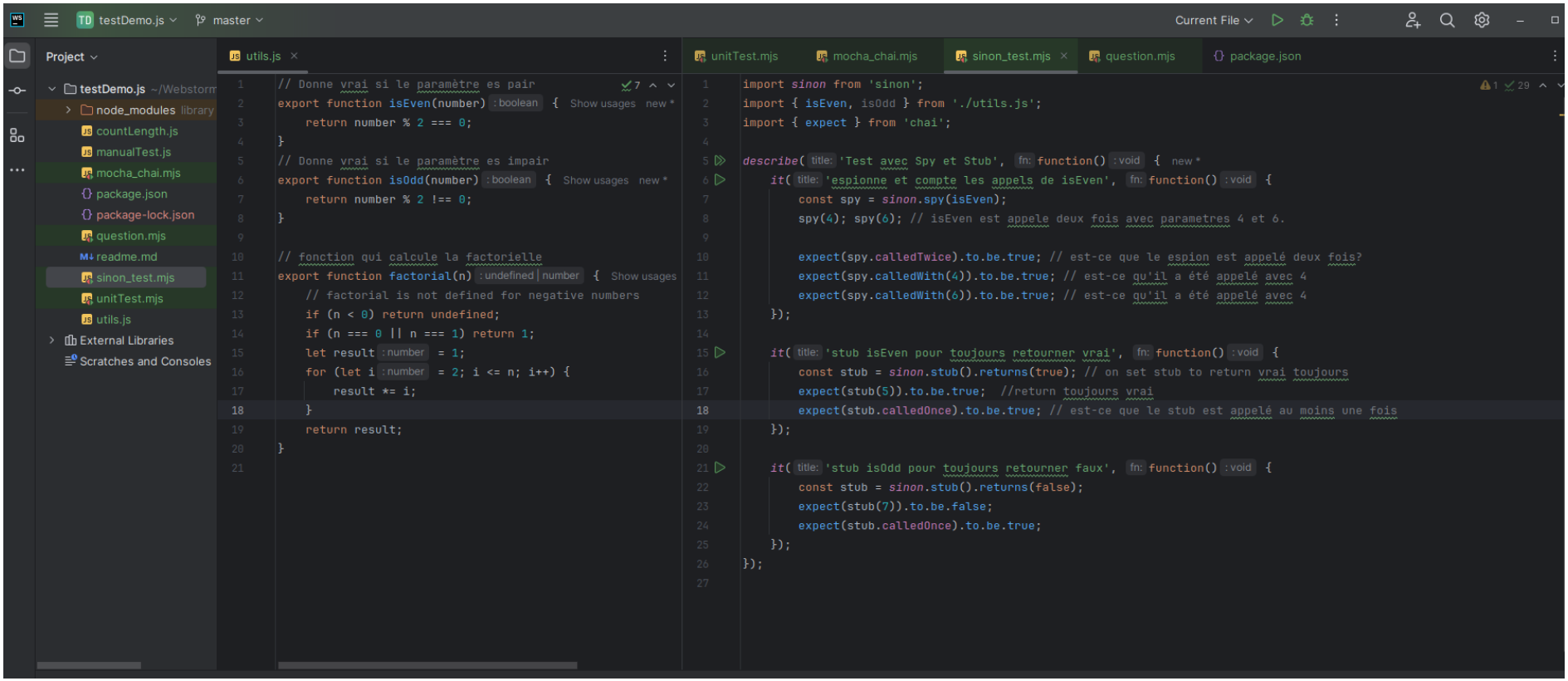


Figure: Exemple Sinon

Université de Montréal

Projet personnel Demo

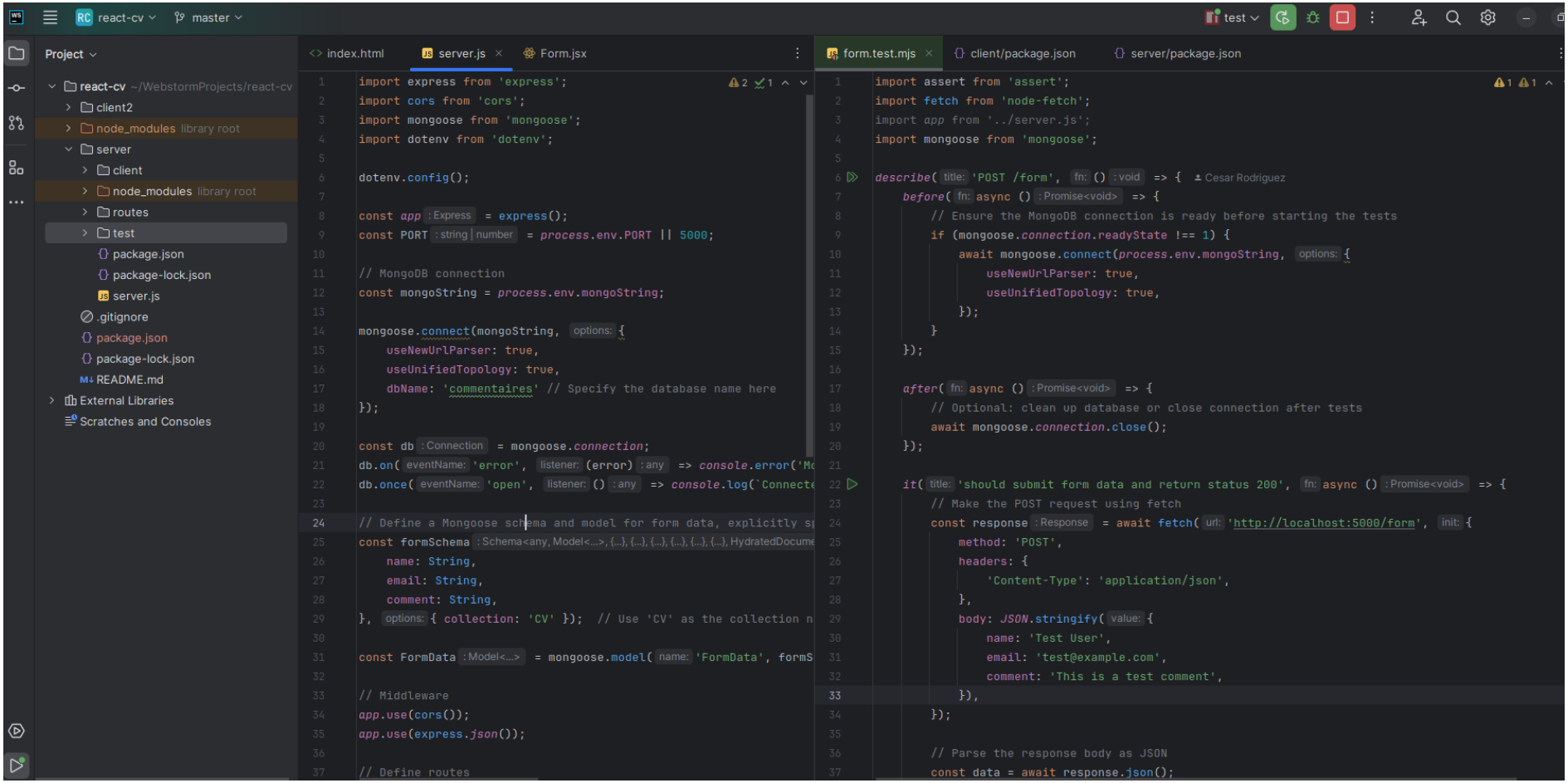


Figure: Projet personnel

Université de Montréal

Conclusion

- Les test unitaires sont la base de la conception TDD.
- Les test unitaires servent à vérifier la bonne qualité du code et à prevenir 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

Université de Montréal