Documentation -
Mettre à disposition des utilisateurs un service informatique

Réaliser les tests d'intégration et d'acceptation d'un service

BTS - 2023

Informations

Nom du projet	TEST UNITAIRE
Type de document	Document
Mots-clés	bts sio- options slam - NodeJS - Jest - test unitaire - BDD - mariadb - backend
Auteur	Laura Paillet
Point(s) du tableau de synthèse	Réaliser les tests d'intégration et d'acceptation d'un service

Table des matières

1 - Résumé du document

Pour le contexte de cette documentation, elle fait suite à un TP effectué en cours, dans lequel j’ai dû créer une BDD mariadb avec les tables “utilisateur” et “recette”. Par la suite, j'ai mis en place un backend en NodeJS. Puis j’ai effectué des tests unitaires avec Jest.

J’expliquerais ce qu’est Jest et supertest, puis quels outils seront nécessaire à installer avec npm et enfin nous effectuerons les tests unitaires sur les CRUD (Create, Read, Update, Delete).

2 - Qu’est-ce Jest et supertest?

- Jest :

Jest est un framework de test JavaScript open-source. Il est conçu pour tester des applications JavaScript avec des fonctionnalités telles que la prise en charge des tests unitaires, d'intégration, de snapshot et des tests de performance. La syntaxe de Jest est simple et facile à comprendre, ce qui facilite son utilisation pour les développeurs.

Jest offre également différentes fonctionnalités telles que la détection automatique des tests, la parallélisation des tests et la génération de rapports de couverture de code. Il est également facile à configurer avec des outils tels que Babel, Webpack et TypeScript.

- Supertest:

SuperTest est une bibliothèque de tests pour les applications Node.js qui s'intègre avec Jest pour permettre des tests d'API HTTP simples et efficaces.

En utilisant SuperTest avec Jest, vous pouvez facilement effectuer des tests d'API pour vérifier si les endpoints de votre application fonctionnent correctement.
Par exemple, vous pouvez tester si une requête GET renvoie le résultat attendu, ou si une requête POST insère des données dans la base de données de manière appropriée.

SuperTest est facile à configurer et à utiliser, en utilisant SuperTest avec Jest, vous pouvez automatiser vos tests d'API et vous assurer que votre application fonctionne correctement à chaque modification.

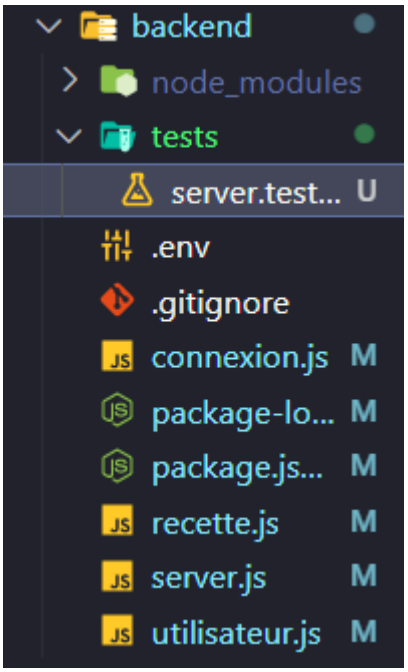
3 - Installation d'outils nécessaires aux tests unitaires

Pour installer Jest et supertest sur son projet, nous utiliserons des commandes npm (Node Package Manager):

- npm install --save-dev jest
- npm install jest supertest --save-dev

On peut vérifier que les dépendances se soient bien installé dans le package.json.

Puis une fois les pages server.js / utilisateur.js / recette.js créé, on va créer un fichier "tests" et dans ce fichier, on va pour nos tests, créer un fichier "server.test.js".



Dans ce fichier server.test il nous faudra importer server.js (dans lequel on aura préalablement mis la possibilité d'exportation).

Fichier server.js :

```
module.exports = app

app.listen(7000, function () {
    console.log('CORS-enabled web server listening on port 7000');
})
```

Fichier server.test.js:

```
const request = require('supertest');
const app = require('../server');
```

4 - Test CRUD

Par la suite on peut commencer les tests CRUD (Create, Read, Update, Delete).
Pour ma première requête GET j'ai eu une erreur, à la suite de mon "npm test" pour démarrer le test unitaire.

```
FAIL tests/server.test.js
  x GET /utilisateur (223 ms)

  ● GET /utilisateur

    expect(received).toBe(expected) // Object.is equality

    Expected: 200
    Received: 404

       7 |     let res = await request(app)
       8 |         .get('/utilisateur');
    >  9 |         expect(res.statusCode).toBe(200)
         |                                ^
      10 |         expect(res._body).toBeDefined();
      11 |     console.log(res);
      12 | })

      at Object.toBe (tests/server.test.js:9:32)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        1.614 s, estimated 2 s
Ran all test suites.
```

L'origine de l'erreur provient de mon code dans le test unitaire, où je n'ai pas spécifier l'id de l'utilisateur:

```
ckend > tests > server.test.js > it('GET /utilisateur') callback > l
1   const request = require('supertest');
2   const app = require('../server');
3
4
5   it('GET /utilisateur', async () => {
6       let res = await request(app)
7           .get('/utilisateur');
8           expect(res.statusCode).toBe(200)
9           expect(res._body).toBeDefined();
10  })
11
```

Pour corriger cette erreur, j'ai rajouté l'id comme ci-dessous:

```
it('GET /utilisateur', async () => {
    let res = await request(app)
        .get('/utilisateur/1');
        expect(res.statusCode).toBe(200)
        expect(res._body).toBeDefined();
})
```

Et le test s'effectue correctement:

```
      type:  application/json ;
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.388 s, estimated 2 s
Ran all test suites.
```

- Test GET pour la table recette:

```
it('GET /recette', async () => {
    let res = await request(app)
        .get('/recette/1');
        expect(res.statusCode).toBe(200)
        expect(res._body).toBeDefined();
})
```

```
PS C:\Users\laura\Documents\SLAM\react\tpTestUnitaire\backend> npm test

> backend@1.0.0 test

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.917 s, estimated 2 s
Ran all test suites.
```