16/03/2023

Documentation Test Unitaire Jest et Supertest

IPSSI MLV

MAXIME CHENET
BTS SIO SLAM

Table des matières

Définitions **2**

Définition de Jest 2

Définition de supertest 2

Préparer ses tests **2**

Création du dossier 2

Création du fichier 2

Installation du fichier de test 2

Installation des packages 2

Vérifier package.json 3

Utiliser Jest **3**

Describe() 3

It() 3

Expect() 3

Importer les modules 4

Test des routes « /cinema/ » 4

Test des routes « /film/ » 5

Lancer le test **5**

npm test 5

Définitions

Définition de Jest

Jest est un framework créé par Facebook utilisé pour le test dans des projets Javascript. Il est simple à utiliser grâce à ses noms de fonction intuitives et facile à comprendre.

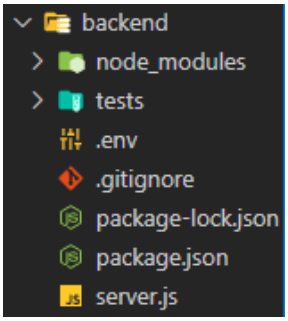
Définition de supertest

Supertest est une bibliothèque de tests d'intégration pour Node.js simulant des requêtes http. Celle-ci permet notamment de tester les routes d'une application Express

Préparer ses tests

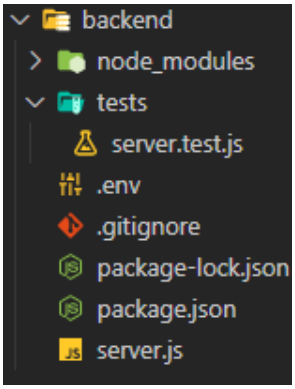
Création du dossier

Dans le projet, créez un nouveau dossier à la racine nommé « tests » :



Création du fichier de test

Dans ce dossier, créez un fichier qui servira pour faire vos tests. Il est conseillé de le nommer en fonction de votre fichier js que vous voulez tester, de plus, il doit se terminer par «.test.js». Par exemple « server.test.js »



Installation des packages

Vous devez maintenant installer 2 packages : Jest et supertest. Tapez les commandes suivantes à la racine de votre projet :

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

```
npm install supertest --save-dev
```

Vérifier package.json

Il est important de vérifier le fichier « package.json ». Modifiez le texte dans « test » en le remplaçant par « jest » comme ceci :

```
{
  "name": "cinema",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  ▷ Debug
  "scripts": {
    "test": "jest",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "mariadb": "^3.0.2"
  },
  "devDependencies": {
    "jest": "^29.5.0",
    "supertest": "^6.3.3"
```

Utiliser Jest

Describe()

```
describe('Test des routes Ville', () => {

})
```

It()

Cette fonction correspond à ce que l'on attend du programme s'il fonctionne :

```
describe('Test des routes Ville', () => {
    it('Get All ville', () => {

    });
})
```

Expect()

Cette fonction correspond aux conditions pour que le test soit réussi. Dans cet exemple, on souhaite que le statut soit un code 200 :

```
describe('Test des routes Ville', () => {
    it('Get All ville', () => {
        const response = request(app).get('/ville');
        expect(response.status).toBe(200);
    });
})
```

Importer les modules

A la fin de votre fichier server.js, vous devez exporter votre application Express afin de la récupérer dans notre fichier de test :

```
module.exports = app;
```

Vous la récupérez au début de votre fichier de test, avec require(). Profitez-en pour importer également « supertest »

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

Test des routes « /cinema/ »

Voici un exemple de test pour la route cinéma :

```
describe("Test des routes Cinéma", () => {
    it('Get all cinema', async () => {
        const response = await request(app).get('/cinema');
        expect(response.status).toBe(200);
        expect(response.body).toEqual([
        { id: 1, nom: 'Pathé Melun', adresse: '34 rue saint aspais', idVille:
2 },
        { id: 2, nom: 'Le cinéma de montcuq', adresse: "16 impasse de
l'arrière", idVille: 1 }
        ]);
    });
})
```

Nous attendons du test que la réponse soit un code 200 ainsi que les données retournées soit 2 objets comportant un id, un nom, une adresse et un idVille.

Test de la route « /film/ »

Dans cet exemple, on souhaite tester la route permettant de mettre à jour un film existant dans la base de données :

```
describe("Test des routes Films", () => {
    it("Put film", async () => {
        const testFilm = {
            id: 9,
            titre: 'testFilmUpdate',
            duree: 120

        };
        const response = await request(app)
            .put('/film/')
            .send(testFilm)
            .expect(200);
        expect(response.status).toBe(200);

        expect(response.body).toContainEqual(expect.objectContaining(testFilm)
);

    })
})
```

Lancez le test pour vérifier si tout est correct.

Lancer un test

npm test

Pour lancer un test, vous devez taper la commande « npm test » :

```
npm test
```

Résultat du test :

```
Test Suites: 1 failed,
Tests:       1 failed,
Snapshots:   0 total
Time:        2.361 s,
Ran all test suites.
```

Il semblerait que le test ait échoué. Après revérification du code et de la route, on se rend compte que cette route attend l'id du film à modifier.

On modifie alors de cette manière :

```
describe("Test des routes Films", () => {
    it("Put film", async () => {
        const testFilm = {
            id: 9,
            titre: 'testFilmUpdate',
            duree: 120

        };
        const response = await request(app)
            .put('/film/' + testFilm.id)
            .send(testFilm)
            .expect(200);
        expect(response.status).toBe(200);

        expect(response.body).toContainEqual(expect.objectContaining(testFilm));

    })
})
```

Relançons le test :

```
Test Suites: 1 passed,
Tests:       2 passed,
Snapshots:   0 total
Time:        2.366 s, e
```

Le test est maintenant correct.