Développement des tests unitaires sur une application Node.js avec Jest et SuperTest

Jest est un framework de test JavaScript, Il permet d'écrire et d'exécuter des tests unitaires.

SuperTest est une librairie qui sert à tester les APIs écrites en Node.js

Nous avons utilisé les tests unitaire sur l'API REST, lié à une base de donnée de cinémas, faite sur NodsJS afin de vérifier que toutes les routes fonctionnes bien

Table cinema de la base de donnée cinema

id	nom	adresse	idVille
1	Pathé Melun	34 rue saint aspais	2
2	Le cinéma de Lyon	16 impasse de l'arrière	1

Table film de la base de donnée cinema

id	titre	duree
1	les dents de la mer	120
2	Le seigneur des oignons	120

Voici comment nous avons connecté la base de donnée à l'API

server.js

```
const mariadb = require('mariadb')
const express = require('express')
const app = express()
var cors = require('cors')

require('dotenv').config()

const pool = mariadb.createPool({
    host: process.env.DB_HOST,
    database: process.env.DB_DTB,
    user: process.env.DB_USER,
    password: process.env.DB_PWD
})
```

```
app.listen(8000, () =>{
    console.log("Serveur à l'écoute");
} )

module.exports = app
```

.env

```
DB_HOST= 'localhost'
DB_DTB= 'cinema'
DB_USER= 'root'
DB_PWD= ''
```

Et puis voici comment nous avons installer Jest et SuperTest

Dans le terminal :

```
npm i jest --save-dev
```

```
npm i supertest --save-dev
```

Premièrement nous allons vérifier la première route, qui permet d'afficher les cinémas. Nous vérifions si le code de retour est égal à 200, si le résultat est définit et si le contenus de la réponse est bien égal à celle de la base de donnée

```
app.get('/cinema', async(req, res) =>{
    let conn;
    conn = await pool.getConnection();
    const rows = await conn.query('SELECT * FROM cinema;')
    res.status(200).json(rows)
})
```

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

```
PASS test/server.test.js
  Test server.js
    √ Test: affichage des cinemas (114 ms)
    √ Test: modification d'un cinema (52 ms)
    √ Test: supression d'un film (20 ms)
    √ Test: ajout d'un film (15 ms)

A worker process has failed to exit gracefully
ndles to find leaks. Active timers can also cau
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
```

Dans cette situation nous voyons que le même test a renvoyé « fail », effectivement le nom du cinéma n’est pas le bon. Pour régler cela nous changeons le nom dans la base de donnée

```
● Test server.js › Test: affichage des cinemas

  expect(received).toEqual(expected) // deep equality

  - Expected  - 1
  + Received  + 1

  @@ -7,8 +7,8 @@
      },
      Object {
        "adresse": "16 impasse de l'arrière",
        "id": 2,
        "idVille": 1,
  -     "nom": "Le cinéma de Lyon",
  +     "nom": "Le cinéma de Lyonne",
      },
    ]

     7 |         expect(response.status).toBe(200);
     8 |         expect(response.body).toBeDefined();
   > 9 |         expect(response.body).toEqual([
       |                               ^
    10 |             { id: 1, nom: 'Pathé Melun', adresse: '34 rue saint aspais', idVille: 2 },
    11 |             { id: 2, nom: 'Le cinéma de Lyon', adresse: "16 impasse de l'arrière", idVille: 1 }
    12 |         ]);

    at Object.toEqual (test/server.test.js:9:31)

A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests lea
ndles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.
Test Suites: 1 failed, 1 total
Tests:       1 failed, 3 passed, 4 total
```

```
FAIL test/server.test.js
  Test server.js
    × Test: affichage des cinemas (131 ms)
    √ Test: modification d'un cinema (54 ms)
    √ Test: supression d'un film (20 ms)
    √ Test: ajout d'un film (15 ms)
```

id	nom	adresse	idVille ▾ 1
1	Pathé Melun	34 rue saint aspais	2
2	Le cinéma de Lyonne	16 impasse de l'arrière	1

id	nom	adresse	idVille ▾ 1
1	Pathé Melun	34 rue saint aspais	2
2	Le cinéma de Lyon	16 impasse de l'arrière	1

Maintenant nous allons vérifier la route qui permet de modifier les cinémas, nous envoyons donc une requête pour modifier le cinéma de id 1, puis nous vérifions alors si le réponse est un tableau qui contient en premier le cinéma de id 1, puis nous vérifions si le nom du cinéma a bien changé, et finalement nous remettons grâce à une requête le nom du cinéma de base

```
PASS test/server.test.js
  Test server.js
    √ Test: affichage des cinemas (114 ms)
    √ Test: modification d'un cinema (52 ms)
    √ Test: supression d'un film (20 ms)
    √ Test: ajout d'un film (15 ms)

A worker process has failed to exit gracefully
ndles to find leaks. Active timers can also cau
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
```

```
app.put('/cinema/:id',async(req,res)=>{
    let conn;
    let id = parseInt(req.params.id)
    conn = await pool.getConnection();
    await conn.query(`UPDATE cinema SET nom = ?, adresse = ?, idVille = ? WHERE id = ${id}`,[req.body.nom,req.body.adresse,req.body.idVille])
    const rows = await conn.query('SELECT * FROM cinema;')
    res.status(200).json(rows)
})
```

```
it("Test: modification d'un cinema", async ()=>{
    await request(app).put('/cinema/1').send({nom: "Pathé Disney", adresse: '34 rue saint aspais', idVille: 2});
    const response = await request(app).get('/cinema/1');
    expect(response.body).toEqual(expect.any(Array));
    expect(response.body[0].id).toBe(1);
    expect(response.body[0].nom).toBe("Pathé Disney");
    await request(app).put('/cinema/1').send({nom: "Pathé Melun", adresse: '34 rue saint aspais', idVille: 2});
})
```

Dans ce test nous allons vérifier la route de suppression d'un film en envoyant une requête afin de supprimer le film de id 1, puis nous vérifions si le code de réponse est bien 200, ce qui signifie que la surpression a bien fonctionné, puis nous remettons le film précédemment supprimé.

```
PASS test/server.test.js
  Test server.js
    √ Test: affichage des cinemas (114 ms)
    √ Test: modification d'un cinema (52 ms)
    √ Test: supression d'un film (20 ms)
    √ Test: ajout d'un film (15 ms)

A worker process has failed to exit gracefully
ndles to find leaks. Active timers can also cau
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
```

```
app.delete('/film/:id', async(req, res) =>{
    let conn;
    let id = parseInt(req.params.id)
    conn = await pool.getConnection();
    await conn.query(`DELETE FROM film WHERE id = ?;`,[id])
    const rows = await conn.query(`SELECT * FROM film;`)
    res.status(200).json(rows)
})
```

```
it("Test: supression d'un film", async ()=>{
    const response = await request(app).delete('/film/1');
    expect(response.status).toBe(200);
    await request(app).post('/film2').send({id: 1, titre: "les dents de la mer", duree: 120});
})
```

Dans ce dernier test nous vérifions la route d'ajout d'un film, nous faisons donc l'inverse de la précédente route en ajoutant un nouveau film, en vérifiant le code de réponse puis en supprimant le film précédemment créé

```
PASS test/server.test.js
  Test server.js
    √ Test: affichage des cinemas (114 ms)
    √ Test: modification d'un cinema (52 ms)
    √ Test: supression d'un film (20 ms)
    √ Test: ajout d'un film (15 ms)

A worker process has failed to exit gracefully
ndles to find leaks. Active timers can also cau
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
```

```
app.post('/film2',async(req,res)=>{
    let conn;
    conn = await pool.getConnection();
    await conn.query('INSERT INTO film(id, titre, duree) VALUES (?,?,?)',[req.body.id, req.body.titre,req.body.duree])
    const rows = await conn.query('SELECT * FROM film;')
    res.status(200).json(rows)
})
```

```
it("Test: ajout d'un film", async ()=>{
    const response = await request(app).post('/film2').send({id: 3, titre: "Avengers", duree: 134});
    expect(response.status).toBe(200);
    await request(app).delete('/film/3');
})
```