



# Documentation

Mise en place de tests unitaires  
sur une application NodeJs

Valide le point : Réaliser les tests  
d'intégration et d'acceptation d'un  
service

Loan ALBIACH  
ECOLE IPSSI (MARNE-LA-VALLEE)

## Présentation du projet :

Nous avons créé un backend NodeJs sur le thème du cinéma,  
une base de données possédant les films et les cinémas a été créé.

Différentes routes permettant de récupérer, modifier, supprimer et ajouter des cinémas  
ainsi que des films ont été développées.

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

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

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

app.put('/film/:id', async(req, res) =>{
  let conn;
  let id = parseInt(req.params.id)
```


Une liaison à la base de données a été effectué grâce au code suivant :

```
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
})
```

La constante « pool » contient les identifiants de connexion à la base de données  
que nous allons réutiliser dans les routes.



Les tests unitaires sont des tests qui vérifient que chaque unité de code (fonction, méthode, module, etc.) fonctionne comme prévu. Les tests unitaires sont souvent automatisés pour faciliter la maintenance du code et pour garantir que les modifications apportées ne perturbent pas les fonctionnalités existantes.

Pour faire des tests unitaires, nous devons installer Jest et Supertest.

```
PS C:\Users\-----\Desktop\Ecole\2eme annee\SLAM\NodeJS\Cinema\backend> npm install --save-dev jest supertest
```

Une fois installé, il faut modifier la partie script du package.json

```
"scripts": {  
  "test": "jest",  
  "start": "node server.js"  
},
```

Nous devons ensuite exporter les routes présentes dans le fichier server.js grâce à cette ligne de code :

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

Puis, dans le fichier server.test.js, nous importons supertest (permet d'effectuer des requêtes http sur une application express) dans la variable « request » et le fichier « server.js » dans la variable « app ».

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

Une fois toutes ces étapes effectuées, il suffit de faire la commande « npm test » pour démarrer un test unitaire.

Voici le code que nous allons tester en premier, il permet d'obtenir la liste des cinémas présent dans la base de données, dans la table « cinema ».

```
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)
})
```

Ci-dessous, le code permettant de tester si l'on récupère bien la liste des cinémas présents dans la base de données.

La variable « response » possède la liste des cinémas de la base de données

La ligne « expect(response.status).toBe(200) ; » permet de vérifier s'il y a aucune erreur.

La ligne « expect(response.body).toEqual([...]) ; » permet de vérifier si la variable « response » est égal à ce qu'il y a entre les [...]

```
describe('Test endpoint /cinema', () => {
  it('Récupérer les cinémas', 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 }
    ]);
  });
});
```

Pour lancer le test, effectuez la commande « npm test ». Le résultat obtenu est le suivant :

```
PASS tests/server.test.js
  Test endpoint /cinema
    ✓ Récupérer les cinémas (59 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.022 s
```

Dans ce cas, les tests passés sont vérifiés car ils apparaissent en vert.

En contre-exemple, voici un test qui n'est pas vérifié :

```
describe('Test endpoint /cinema', () => {
  it('Récupérer les cinémas', 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: 52 },
      { id: 2, nom: 'Le cinéma de montcuq', adresse: "16 impasse de l'arrière", idVille: 1 }
    ]);
  });
});
```

Le résultat de ce test est :

```
Test Suites: 1 failed, 1 total
Tests:      1 failed, 1 total
Snapshots:  0 total
Time:       0.848 s, estimated 1 s
Ran all test suites.
```

Plus précisément, on peut apercevoir quel(s) est(sont) les problèmes, pourquoi le test n'est pas vérifié :

```
-   "idVille": 52,
+   "idVille": 2,
```

Dans ce cas, nous remarquons que l'idVille du cinéma « Pathé Malun » est 52, or, dans la base de données, l'idVille est 2.

Nous allons effectuer un second test unitaire qui lui, permet d'ajouter un film à la base de données :

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

Dans un premier temps, on ajoute le film à la base de données grâce à la ligne `.send(...)` en mettant en paramètre les valeurs demandées dans la route présente sur la capture d'écran ci-dessus.

Une fois ajouté, nous testons si le film à bien été ajouté dans la base de données et s'il y a eu une erreur.

```
it('Ajouter un film /film', async () => {
  const response = await request(app).post('/film').send({titre: "L'age de glace 1", duree: 132});
  expect(response.status).toBe(200);
  expect(response.body[6]).toEqual({"duree": 132, "id": 7, "titre": "L'age de glace 1"});
});
```

Nous pouvons observer aucunes erreurs dans ce test unitaire :

```
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.852 s, estimated 1 s
```

Nous apercevons 2 tests passés sur 2 car j'ai mis les 2 tests à la suite :

```
describe('Test unitaire', () => {
  it('Récupérer les cinémas', 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 }
    ]);
  })
  it('Ajouter un film /film', async () => {
    const response = await request(app).post('/film').send({titre: "L'age de glace 1", duree: 132});
    expect(response.status).toBe(200);
    expect(response.body[6]).toEqual({"duree": 132, "id": 7, "titre": "L'age de glace 1"});
  });
});
```

Si dans le test, je remplace la vérification de « L'age de glace 1 » par « L'age de glace 2 », nous verrons qu'il y a une erreur :

```
it('Ajouter un film /film', async () => {  
  const response = await request(app).post('/film').send({titre: "L'age de glace 1", duree: 132});  
  expect(response.status).toBe(200);  
  expect(response.body[6]).toEqual({"duree": 132, "id": 7, "titre": "L'age de glace 2"});  
});
```

```
Test Suites: 1 failed, 1 total  
Tests:      1 failed, 1 passed, 2 total  
Snapshots:  0 total  
Time:       0.899 s, estimated 1 s
```

```
-  "titre": "L'age de glace 2",  
+  "titre": "L'age de glace 1",
```