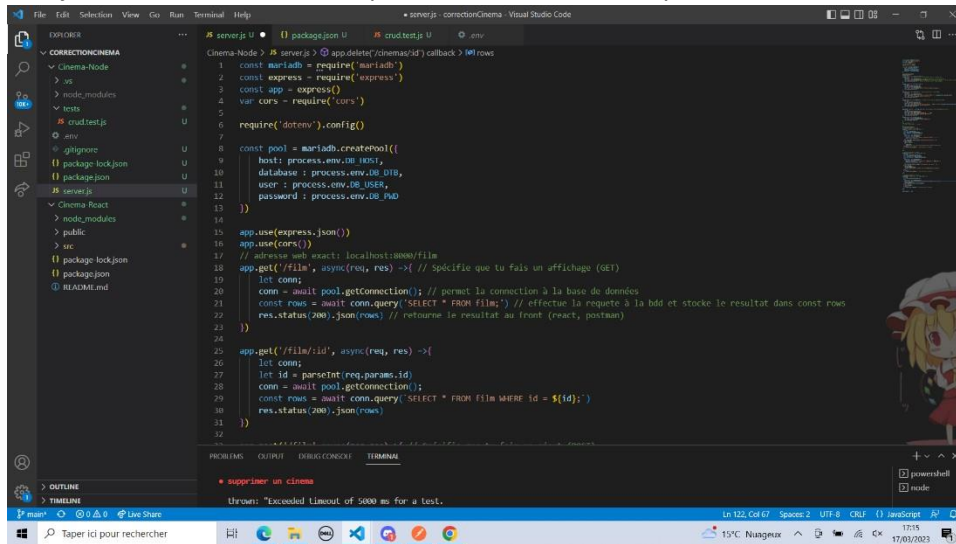
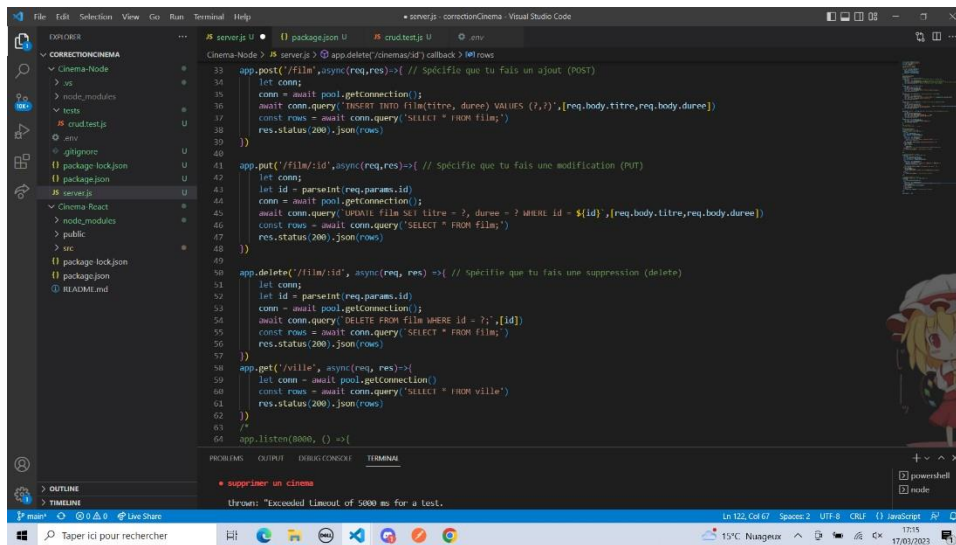


# Test unitaire

J'ai ajouté 3 routes différentes pour la table cinéma, en plus des routes existantes :



```
1 const mariadb = require('mariadb')
2 const express = require('express')
3 const app = express()
4 var cors = require('cors')
5 require('dotenv').config()
6
7 const pool = mariadb.createPool({
8   host: process.env.DB_HOST,
9   database: process.env.DB_DB,
10   user: process.env.DB_USER,
11   password: process.env.DB_PWD
12 })
13
14 app.use(express.json())
15 app.use(cors())
16
17 // route pour récupérer la liste des films
18 app.get('/film', async (req, res) => { // Spécifie que tu fais un affichage (GET)
19   let conn;
20   conn = await pool.getConnection(); // permet la connexion à la base de données
21   const rows = await conn.query('SELECT * FROM film;') // effectue la requête à la bdd et stocke le résultat dans const rows
22   res.status(200).json(rows) // retourne le résultat au front (react, postman)
23 })
24
25 // route pour récupérer un film par id
26 app.get('/film/:id', async (req, res) => {
27   let conn;
28   let id = parseInt(req.params.id)
29   conn = await pool.getConnection();
30   const rows = await conn.query('SELECT * FROM film WHERE id = ${id};')
31   res.status(200).json(rows)
32 })
33
34 // route pour supprimer un film
35 app.delete('/cinema/:id', async (req, res) => {
36   let conn;
37   let id = parseInt(req.params.id)
38   conn = await pool.getConnection();
39   const rows = await conn.query('DELETE FROM film WHERE id = ${id};')
40   res.status(200).json(rows)
41 })
42
43 app.listen(3000, () => {
44   console.log('Le serveur est en écoute sur le port 3000')
45 })
```



```
33 app.post('/film', async (req, res) => { // Spécifie que tu fais un ajout (POST)
34   let conn;
35   conn = await pool.getConnection();
36   const rows = await conn.query('INSERT INTO film (titre, duree) VALUES (?, ?)', [req.body.titre, req.body.duree])
37   const rows = await conn.query('SELECT * FROM film;')
38   res.status(200).json(rows)
39 })
40
41 // route pour modifier un film
42 app.put('/film/:id', async (req, res) => { // Spécifie que tu fais une modification (PUT)
43   let conn;
44   let id = parseInt(req.params.id)
45   conn = await pool.getConnection();
46   const rows = await conn.query('UPDATE film SET titre = ?, duree = ? WHERE id = ${id};', [req.body.titre, req.body.duree])
47   const rows = await conn.query('SELECT * FROM film;')
48   res.status(200).json(rows)
49 })
50
51 // route pour supprimer un film
52 app.delete('/film/:id', async (req, res) => { // Spécifie que tu fais une suppression (DELETE)
53   let conn;
54   let id = parseInt(req.params.id)
55   conn = await pool.getConnection();
56   const rows = await conn.query('DELETE FROM film WHERE id = ${id};')
57   const rows = await conn.query('SELECT * FROM film;')
58   res.status(200).json(rows)
59 })
60
61 // route pour récupérer un film par id
62 app.get('/film/:id', async (req, res) => {
63   let conn;
64   let id = parseInt(req.params.id)
65   conn = await pool.getConnection();
66   const rows = await conn.query('SELECT * FROM film WHERE id = ${id};')
67   res.status(200).json(rows)
68 })
69
70 app.listen(3000, () => {
71   console.log('Le serveur est en écoute sur le port 3000')
72 })
```

```

// Route pour ajouter un cinéma
app.post('/cinemas', async (req, res) => {
  let conn;
  const nom = req.body.nom;
  const adresse = req.body.adresse;
  const codePostal = req.body.codePostal;
  try {
    conn = await pool.getConnection();
    const rows = await conn.query(
      "INSERT INTO cinema (nom, adresse, codePostal) VALUES (?, ?, ?)",
      [nom, adresse, codePostal]
    );
    res.status(200).json({ message: "cinéma ajouté avec succès !" });
  } catch (err) {
    console.log(err);
    res.status(500).json({ error: "Une erreur est survenue lors de l'ajout du cinéma." });
  } finally {
    if (conn) conn.release(); // libération de la connexion
  }
});

// Route pour modifier un cinéma
app.put('/cinemas/:id', async (req, res) => {
  let conn;
  const id = parseInt(req.params.id);
  const nomCinema = req.body.nomCinema;
  const adresseCinema = req.body.adresseCinema;
  try {
    conn = await pool.getConnection();
    const rows = await conn.query(
      "UPDATE cinema SET nom = ?, adresse = ?, codePostal = ?, WHERE id = ?",
      [nomCinema, adresseCinema, id]
    );
    res.status(200).json({ message: "cinéma modifié avec succès !" });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      error: "Une erreur est survenue lors de la modification du cinéma."
    });
  } finally {
    if (conn) conn.release(); // libération de la connexion
  }
});

// Route pour supprimer un cinéma
app.delete('/cinemas/:id', async (req, res) => {
  let conn;
  const id = parseInt(req.params.id);
  try {
    conn = await pool.getConnection();
    const rows = await conn.query("DELETE FROM cinema WHERE id = ?", [id]);
    console.log(rows);
    res.status(200).json(rows.affectedRows);
  } catch (err) {
    console.log(err);
  } finally {
    if (conn) conn.release(); // libération de la connexion
  }
});

module.exports = app;

```

Qu'est-ce qu'un test unitaire ?

Un test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme (appelée « unité » ou « module »).

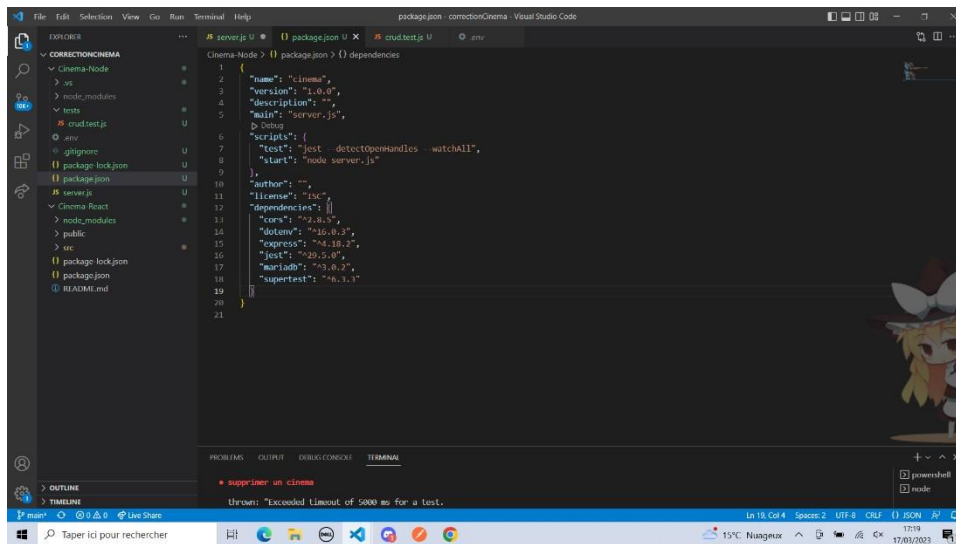
Pour pouvoir réaliser des tests unitaires, nous allons installer supertest avec la commande "npm i supertest"

Nous devons modifier le fichier package.json et ajouter les scripts suivants : jest et node server.js

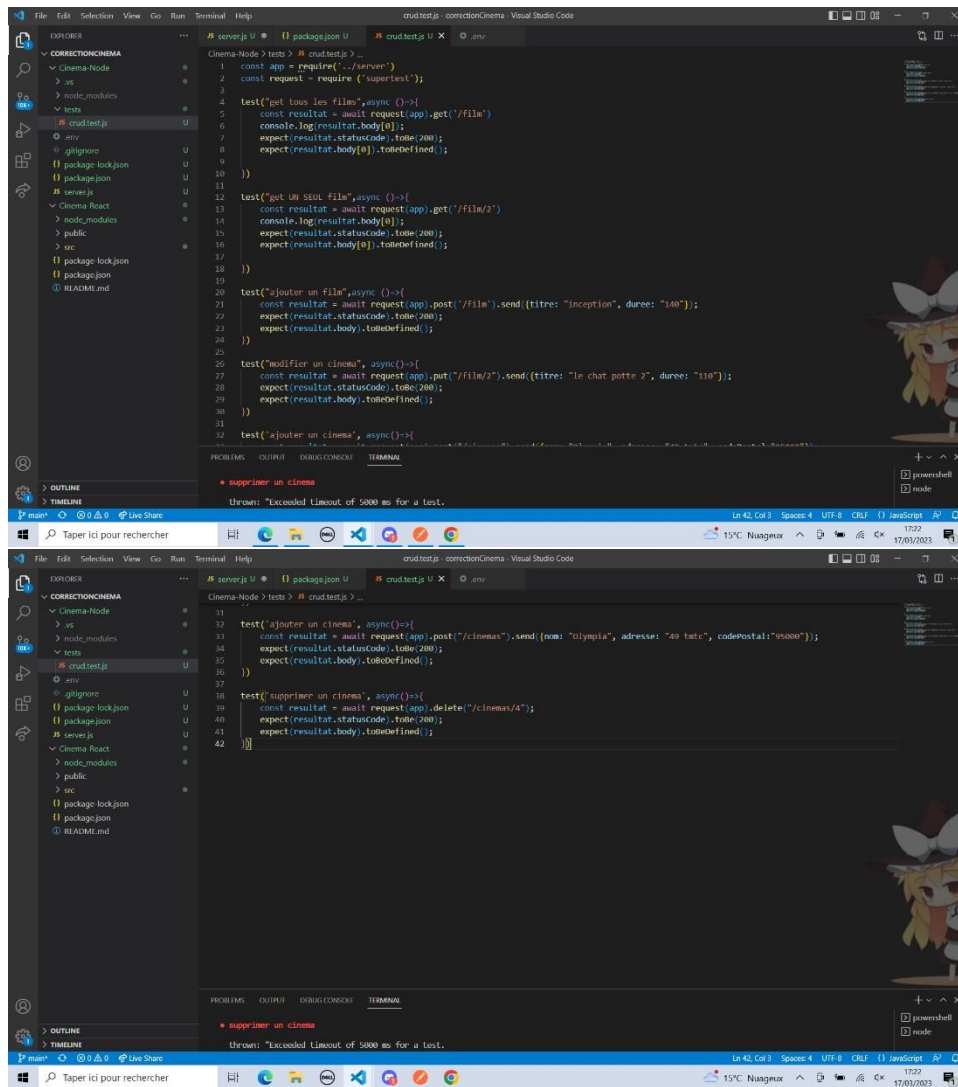
```

{
  "name": "correctionCinema",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "jest",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
    "mysql": "^2.18.0",
    "supertest": "^6.3.3"
  }
}

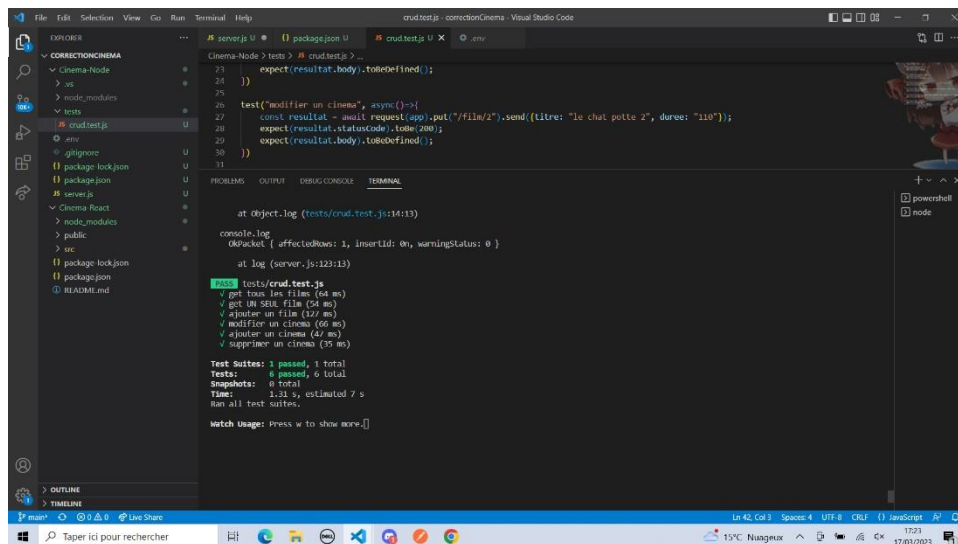
```



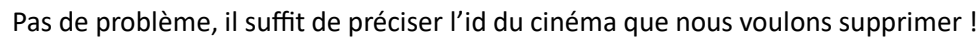
Nous réalisons des tests unitaires sur chacun des CRUD sur des tables de notre base de données, tels que des CREATE, READ, UPDATE et DELETE. Ces tests se trouvent dans un fichier crud.test.js :



Dans le terminal, on écrit la commande “npm test”



Cependant, nous pouvons rencontrer des erreurs, telles que :



Un code 500 correspond à une erreur dans le code, un code 404 correspond à une erreur dans la route.

Un code 200 quant à lui indique que le test a réussi.