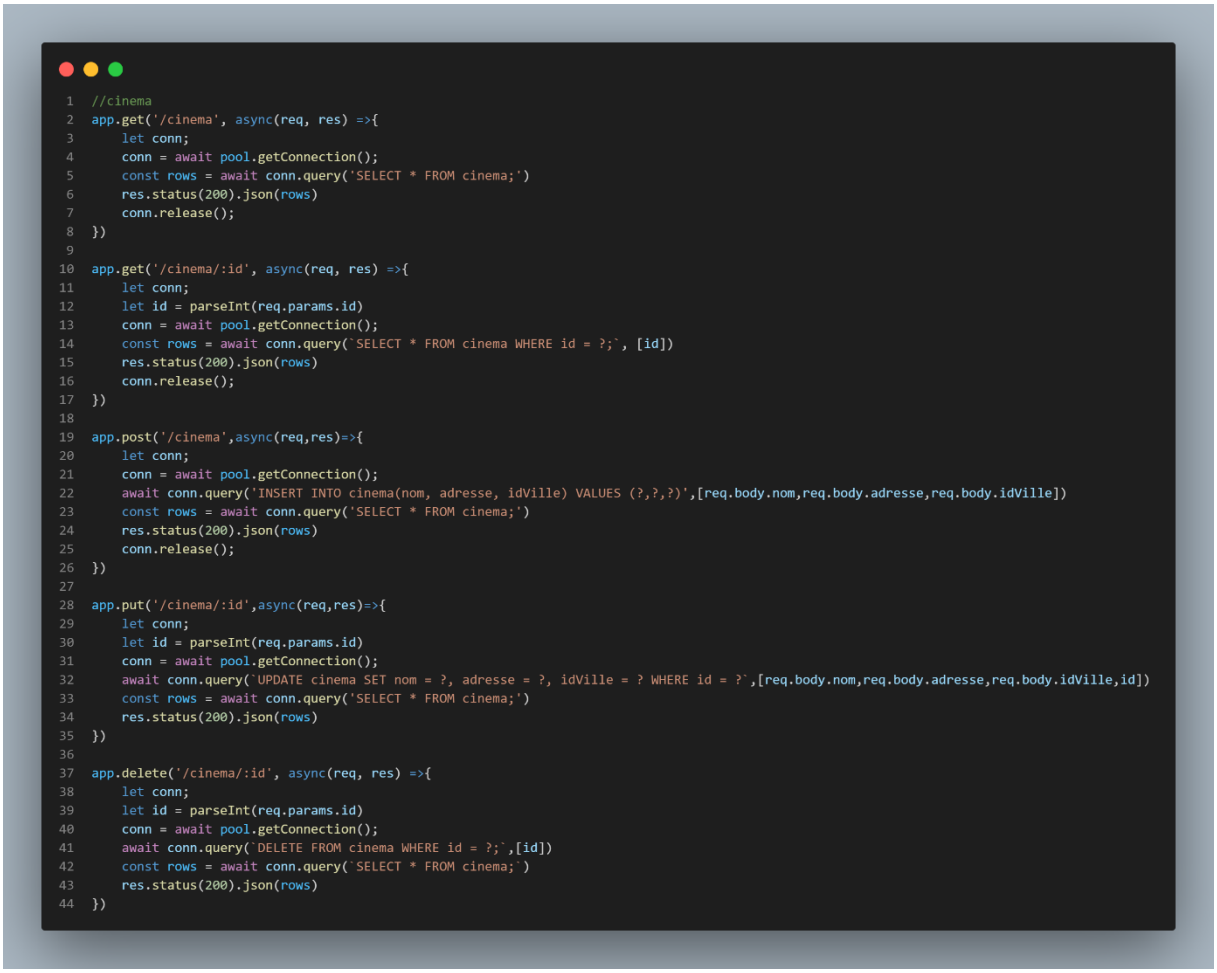
Mettre à disposition des utilisateurs un service informatique
Enzo Espinasse

Après avoir fini un projet nous avons décider de mettre en place des teste unitaire afin de valider le bon fonctionnement de la communication de notre API et la base de donnes.

```
1  const mariadb = require('mariadb')
2  const express = require('express')
3  const app = express()
4  var cors = require('cors')
5
6  require('dotenv').config()
7  const pool = mariadb.createPool({
8      host: process.env.DB_HOST,
9      database : process.env.DB_DTB,
10     user : process.env.DB_USER,
11     password : process.env.DB_PWD
12 })
13
14 app.use(express.json())
15 app.use(cors())
```

Ici nous pouvons voir l'entête de notre fichier « server.js » qui contiens l'ensemble de nos routes. Nous importons mariadb et express qui permette de crée l'API. On initialise ensuite la pool de connections avec les informations stoquer dans un fichier « .env ».

Ici nous pourrons trouver l'ensemble les nouvelles routes lier au CRUD des Cinéma



```
//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)
    conn.release();
})

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

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

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 = ?`,[req.body.nom,req.body.adresse,req.body.idVille,id])
    const rows = await conn.query('SELECT * FROM cinema;')
    res.status(200).json(rows)
})

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

Et pour finir, l'exportations de « app » qui initialise express initialement., nous avons également mis en commentaire les ligne concernant le lancement de l'écoute sur le port 8000 afin de corriger les erreurs avec « Jest » un framework JavaScrypt.

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

module.exports = app
```

Afin d'effectuer nos teste nous avons utilisé Jest, un framework de test JavaScript qui permet de créer des tests unitaires et d'intégration pour les applications. Il fournit une suite de fonctions permettant de réaliser ces tests.

Nous avons également utilisé Supertest qui est une bibliothèque de test pour les applications HTTP. Elle permet de créer des requêtes HTTP pour tester les endpoints de l'application.

En utilisant Supertest avec Jest, il est possible de tester des endpoints de manière automatisée pour vérifier le comportement de l'API.

```
PASS tests/api.test.js
  Test des route ListeCinema
    √ Test de la route [GET] /cinema (34 ms)
    √ Test de la route [GET] /cinema:id (7 ms)
    √ Test de la route [POST] /cinema (25 ms)
    √ Test de la route [PUT] /cinema:id (7 ms)
    √ Test de la route [DELET] /cinema:id (6 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        2.131 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Ici, les 5 route sont donc valide, elle renvoie bien un statu de réponse « 200 », comme escompter dans l'écriture de nos test :

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

describe('Test des route ListeCinema', () => {
    it('Test de la route [GET] /cinema', async () => {
        const res = await request(app).get('/cinema');
        expect(res.statusCode).toBe(200);
    });

    it('Test de la route [GET] /cinema:id', async () => {
        let id = 1;
        const res = await request(app).get(`/cinema/${id}`);
        expect(res.statusCode).toBe(200);
    });

    it('Test de la route [POST] /cinema', async () => {
        let nom = "Cinéma Le Grand Rex";
        let adresse = "1 Boulevard Poissonnière";
        let idVille = "1";
        const res = await request(app).post('/cinema').send({nom, adresse, idVille});
        expect(res.statusCode).toBe(200);
    });

    it('Test de la route [PUT] /cinema:id', async () => {
        let nom = "Cinéma Le Grand Rex";
        let adresse = "1 Boulevard Poissonnière";
        let idVille = "2";
        let id = 3;
        const res = await request(app).put(`/cinema/${id}`).send({nom, adresse, idVille});
        expect(res.statusCode).toBe(200);
    });

    it('Test de la route [DELET] /cinema:id', async () => {
        let id = 7;
        const res = await request(app).delete(`/cinema/${id}`);
        expect(res.statusCode).toBe(200);
    });
});
```