RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

- Les tests unitaires sont une méthode de test de logiciel qui consiste à vérifier le bon fonctionnement d'une unité de code, telle qu'une fonction ou une méthode, de manière isolée et indépendante du reste du programme. L'objectif des tests unitaires est de s'assurer que chaque unité de code fonctionne correctement, en testant toutes les conditions possibles et en détectant les erreurs ou les bogues. Les tests unitaires sont souvent automatisés et peuvent être exécutés de manière régulière tout au long du cycle de développement pour garantir la qualité du code et faciliter la détection et la correction rapide des erreurs.
- C’est ainsi que nous nous sommes essayés à ce procédé à partir d’une API sur le thème du cinéma que nous avons utilisé auparavant. Nous avons fait un CRUD sur la table « cinema » de la base de données afin de faire des tests unitaires sur les routes de ce CRUD que nous verrons par la suite.

RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

- Pour pouvoir faire des test unitaires sur une api on utilise « Jest » et « Supertest » que nous devons installer aux packages :

```
PS C:\wamp64\www\NodeJs> npm i jest supertest
```

- De plus pour pouvoir les utiliser il faut modifier les scripts du projet :

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

- Il faut préciser dans le fichier de test que nous avons besoin de « Supertest » pour tout test sur une API :

```
const request = require('supertest');
```

- Pour que cela fonctionne, il faut également penser à exporte l'application puis l'utiliser dans le fichier test :

Dans l'application API ⟶ `module.exports = app;`

Dans le fichier test ⟶ `const app = require('../Server.js');`

RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

- Après avoir fait tout ceci, nous pouvons passer aux différents tests à réaliser sur chaque route de l'API, voici un exemple de test réaliser :

```
describe('GET /cinema/:id', () => {
    it('should get a single cinema', async () => {
        const res = await request(app).get('/cinema/5');
        expect(res.statusCode).toEqual( expected: 200);
    });
});
```

Cette partie du code teste la récupération d'un seul film avec un id passé en paramètre. Si il reçoit un code « 200 », cela signifie que la route est correcte.

RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

- Il se peut que les tests entrainent des erreurs ce qui signifient que la route ou le test ne sont pas bons :

```
POST /cinema
  × should create a new cinema (37 ms)
```

```
● Test des routes pour le cinéma › POST /cinema › should create a new cinema

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

  Expected: 200
  Received: 500

    31 |                 .post('/cinema')
    32 |                 .send({ prenom: 'Nouveau cinéma', adresse: 'Rue de la nouvelle salle' });
  > 33 |             expect(res.statusCode).toEqual(200);
       |                                    ^
```

- Dans ce cas précis, l'API renvoie un code « 500 » alors que le test attend un code « 200 » en réponse, il faut alors modifier l'erreur pour faire en sorte que le test soit bon.

RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

D’après le code de la route POST :

```
app.post( path: "/cinema", handlers: async (req : … , res : Response<ResBody, LocalsObj> ) => {
    let conn;
    const { nom, adresse } = req.body;
    try {
        conn = await pool.getConnection();
        const rows = await conn.query(
            sql: "INSERT INTO cinema (nom, adresse, idVille) VALUES (?, ?, 1)",
            values: [nom, adresse]
        );
        res.status( code: 200).json( body: { message: "Cinéma ajouté avec succès !" });
    } catch (err) {
        console.log(err);
        res
            .status( code: 500)
            .json( body: { error: "Une erreur est survenue lors de l'ajout du cinéma." });
    } finally {
        if (conn) conn.release(); // Libération de la connexion
    }
});
```

On se rend compte que la route attend des constantes qui sont définis comme « nom » et « adresse », mais dans le test vu précédemment, les variables envoyées s’intitulent « prenom » et « adresse » ce qui entraine forcément une erreur.

RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

Après correction dans le code du test (remplacement de « prenom » par « nom »), il s'avère au final concluant comme est montré ci-dessous :

```
describe('POST /cinema', () => {
    it('should create a new cinema', async () => {
        const res = await request(app)
            .post('/cinema')
            .send({ nom: 'Nouveau cinéma', adresse: 'Rue de la nouvelle salle' });
        expect(res.statusCode).toEqual( expected: 200);
    });
});
```

```
POST /cinema
  √ should create a new cinema (70 ms)
```

RÉALISER LES TESTS D'INTÉGRATION ET D'ACCEPTATION D'UN SERVICE

- Les différentes routes pour le CRUD de la table « cinema » :

```
app.get('/cinema', async (req : … , res : Response<ResBody, LocalsObj> ) => {
    let conn;
    try {
        console.log("lancement de la connexion")
        conn = await pool.getConnection();
        console.log("lancement de la requête")
        const rows = await conn.query( sql: 'SELECT * FROM cinema');
        console.log(rows);
        res.status( code: 200).json(rows);
    } catch (err) {
        console.log(err);
        res.status( code: 500).json(err);
    } finally {
        conn.release();
    }
});

app.get("/cinema/:id", async (req : … , res : Response<ResBody, LocalsObj> ) => {
    let conn;
    const id = parseInt(req.params.id);
    try {
        conn = await pool.getConnection();
        const rows = await conn.query( sql: "SELECT * FROM cinema WHERE id = ?", values: [id]);
        res.status( code: 200).json(rows);
    } catch (err) {
        console.log(err);
    } finally {
        if (conn) conn.release(); // Libération de la connexion
    }
});

// Route pour ajouter un cinéma
app.post( path: "/cinema", handlers: async (req : … , res : Response<ResBody, LocalsObj> ) => {
    let conn;
    const { nom, adresse } = req.body;
    try {
        conn = await pool.getConnection();
        const rows = await conn.query(
            sql: "INSERT INTO cinema (nom, adresse, idVille) VALUES (?, ?, 1)",
            values: [nom, adresse]
        );
        res.status( code: 200).json( body: { message: "Cinéma ajouté avec succès !" });
    } catch (err) {
        console.log(err);
        res
            .status( code: 500)
            .json( body: { 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( path: "/cinema/:id", handlers: async (req : … , res : Response<ResBody, LocalsObj> ) => {
    let conn;
    const id = parseInt(req.params.id);
    const nom = req.body.nom;
    const adresse = req.body.adresse;
    try {
        conn = await pool.getConnection();
        const rows = await conn.query(
            sql: "UPDATE cinema SET nom = ?, adresse = ? WHERE id = ?",
            values: [nom, adresse, id]
        );
        res.status( code: 200).json( body: { message: "Cinéma modifié avec succès !" });
    } catch (err) {
        console.log(err);
        res.status( code: 500).json( body: {
            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( path: "/cinema/:id", handlers: async (req : … , res : Response<ResBody, LocalsObj> ) => {
    let conn;
    const id = parseInt(req.params.id);
    try {
        conn = await pool.getConnection();
        const rows = await conn.query( sql: "DELETE FROM cinema WHERE id = ?", values: [id]);
        console.log(rows);
        res.status( code: 200).json(rows.affectedRows);
    } catch (err) {
        console.log(err);
    } finally {
        if (conn) conn.release(); // Libération de la connexion
    }
});
```