Réaliser les tests d'intégration et d'acceptation d'un service

I. L'application testée

L'application sur laquelle les tests vont être effectuées est une simple application gérant des utilisateurs et des recettes. Elle possède une classe pour chaque avec leurs méthodes respectives.

Son architecture est la suivante :

- Un page servant à la connexion à la base de données

```
const mariadb = require('mariadb')
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
})

module.exports = Object.freeze({
    pool: pool
    });
```

- Une page recette pour la classe du même nom :

```
JS recette.js > ...
1   const { pool } = require('./connexion');
2
3   class recette {
4       constructor( nom, cuisson, difficulte) {
5         this.nom = nom;
6         this.cuisson = cuisson;
7         this.difficulte = difficulte;
8       }
9
10      async getRecette(unId){
11        let conn
12        conn = await pool.getConnection();
13        const rows = await conn.query("SELECT * FROM recette WHERE id = ?", [unId])
14        conn.release();
15        return rows
16      }
17    async addRecette(nom, cuisson, difficulte){
18      let conn
19      conn = await pool.getConnection();
20      await conn.query("INSERT INTO recette(nom, cuisson, difficulte) VALUES (?,?,?) ", [nom, cuisson, difficulte])
21      conn.release();
22      return ('envoie réussi')
23    }
24
25    async delRecette(unId){
26      let conn
27      conn = await pool.getConnection();
28      await conn.query("DELETE FROM recette WHERE id = ?", [unId])
29      conn.release();
30      return ("suppression réussi")
31    }
32
33    async setRecette(unId, nom, cuisson, difficulte){
34      let conn
35      conn = await pool.getConnection();
36      await conn.query("UPDATE recette SET nom = ?, cuisson = ?, difficulte =? WHERE id = ?", [nom, cuisson, difficulte, unId])
37      conn.release();
38      return ("modif réussi")
39    }
40  }
41
42  module.exports = recette;
43
```

- Une page utilisateur pour la classe du même nom :

```
JS utilisateur.js > ...
1   const { pool } = require('./connexion');
2
3   class Utilisateur {
4       constructor(id, nom, prenom, email, mdp) {
5         this.id = id;
6         this.nom = nom;
7         this.prenom = prenom;
8         this.email = email;
9         this.mdp = mdp;
10      }
11
12      async getUtilisateur(unId){
13          let conn
14          conn = await pool.getConnection();
15          const row = await conn.query("SELECT * FROM utilisateur WHERE id = ?", [unId])
16          conn.release();
17          return row
18        }
19      async addUtilisateur(nom,prenom,email,mdp){
20        let conn
21        conn = await pool.getConnection();
22        await conn.query("INSERT INTO utilisateur(nom, prenom, email,mdp) VALUES (?,?,?,?) ", [nom,prenom,email,mdp])
23        conn.release();
24        return ('envoie réussi')
25      }
26
27      async delUtilisateur(unId){
28        let conn
29        conn = await pool.getConnection();
30        await conn.query("DELETE FROM utilisateur WHERE id = ?", [unId])
31        conn.release();
32        return ("suppression réussi")
33      }
34
35      async setUtilisateur(unId, nom, prenom, email, mdp){
36        let conn
37        conn = await pool.getConnection();
38        await conn.query("UPDATE utilisateur SET nom = ?, prenom = ?, email =?, mdp =? WHERE id = ?", [nom, prenom, email, mdp, unId])
39        conn.release();
40        return ("modif réussi")
41      }
42  }
43
44  module.exports = Utilisateur;
45
```

- Une page gérant les routes vers la base données :

Les imports nécessaire, les routes globales ainsi que l'export pour les tests.

```
const express = require('express')
const app = express()
const cors = require('cors')
const { pool } = require('./connexion');
const utilisateur = require("./utilisateur");
const recette = require('./recette')

app.use(cors())
app.use(express.json())

//////// GLOBAL ////////
app.get('/utilisateur', async(req, res) =>{
    let conn;
    console.log('Connexion')
    try{
        conn = await pool.getConnection();
        const rows = await conn.query('SELECT * FROM utilisateur')
        res.status(200).json(rows)
    }catch(err){
        console.log(err)
        throw err;
    }finally{
        if (conn) return conn.end();
    }
})

app.get('/recette', async(req, res) =>{
    let conn;
    console.log('Connexion')
    try{
        conn = await pool.getConnection();
        const rows = await conn.query('SELECT * FROM recette')
        res.status(200).json(rows)
    }catch(err){
        console.log(err)
        throw err;
    }finally{
        if (conn) return conn.end();
    }
})
```

```
module.exports = app

app.listen(8000, function () {
console.log('CORS-enabled web server listening on port 8000')
})
```

-Les routes des méthodes d'Utilisateur

```
//////// UTILISATEUR ////////
app.get("/utilisateur/:id", async (req, res) => {
    let id = parseInt(req.params.id)
    test = new utilisateur()
    let results = await test.getUtilisateur(id)
    res.status(200).json(results)
})

.post("/utilisateur", async (req, res) => {
    test = new utilisateur()
    let results = await test.addUtilisateur(req.body.nom, req.body.prenom, req.body.email, req.body.mdp)
    if(results == undefined){
        console.log("erreur");
    }else{
        console.log(results);
    }
    res.status(200).json(results)
})

.put("/utilisateur/:id", async (req, res) => {
    let id = parseInt(req.params.id)
    test = new utilisateur()
    let results = await test.setUtilisateur(id,req.body.nom, req.body.prenom, req.body.email, req.body.mdp)
    if(results == undefined){
        console.log("erreur");
    }else{
        console.log(results);
    }
    res.status(200).json(results)
})

.delete("/utilisateur/:id", async (req, res) => {
    let id = parseInt(req.params.id)
    test = new utilisateur()
    let results = await test.delUtilisateur(id)
    res.status(200).json(results)
})
```

-Les routes des méthodes de Recette

```
////////// RECETTE //////////
app.get("/recette/:id", async (req, res) => {
    let id = parseInt(req.params.id)
    test = new recette()
    let results = await test.getRecette(id)
    res.status(200).json(results)
})

.post("/recette", async (req, res) => {
    test = new recette()
    let results = await test.addRecette(req.body.nom, req.body.cuisson, req.body.difficulte)
    if(results == undefined){
        console.log("erreur");
    }else{
        console.log(results);
    }
    res.status(200).json(results)
})

.put("/recette/:id", async (req, res) => {
    let id = parseInt(req.params.id)
    test = new recette()
    let results = await test.setRecette(id,req.body.nom, req.body.cuisson, req.body.difficulte)
    if(results == undefined){
        console.log("erreur");
    }else{
        console.log(results);
    }
    res.status(200).json(results)
})

.delete("/recette/:id", async (req, res) => {
    let id = parseInt(req.params.id)
    test = new recette()
    let results = await test.delRecette(id)
    res.status(200).json(results)
})
```

II. Jest et Supertest

Jest est un framework de test de JavaScript développé par Facebook. Il est souvent utilisé pour tester des applications React, mais il peut être utilisé pour tester tout type de projet JavaScript, dans notre cas pour du NodeJs

Jest fournit une interface facile à utiliser pour écrire des tests unitaires, des tests d'intégration et des tests de bout en bout. Il comprend également un ensemble d'outils pour faciliter l'écriture de tests, tels que des fonctions de comparaison d'assertion et des utilitaires de mock.

L'un des avantages de Jest est qu'il inclut également un outil de surveillance intégré, qui peut être configuré pour exécuter des tests automatiquement lorsqu'un fichier est modifié. Cela rend l'écriture de tests plus rapide et plus efficace. En outre, Jest est assez rapide et peut exécuter des tests en parallèle, ce qui accélère les temps d'exécution des tests.

Supertest est une bibliothèque de test de Node.js qui permet de tester les applications Web en effectuant des requêtes HTTP simulées. Elle est souvent utilisée pour tester les API RESTful.

Avec Supertest, vous pouvez simuler une requête HTTP (par exemple, une requête GET ou POST) à votre application Web, puis vérifier la réponse renvoyée. Cela permet de tester votre application de manière complète, en testant non seulement le code, mais également les interactions avec l'interface utilisateur et les services externes.

Supertest est souvent utilisé en conjonction avec des frameworks de test de Node.js tels que Mocha ou Jest. Il fournit une interface facile à utiliser pour effectuer des requêtes HTTP simulées et vérifier les réponses, ainsi que des fonctions pour configurer les en-têtes de requête, les données de formulaire et les cookies.

III. Les tests effectués

Pour commencer, voici une capture d'écran de tous nos tests après correction des erreurs présentés plus loin :

Premièrement les tests pour les méthodes de la classe Utilisateur :

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

describe("Test Utilisateur", () => {
    it('/utilisateur Devrait renvoyer le statut 200 OK', async () => {
        const response = await request(app).get('/utilisateur');
        console.log(response.length);
        expect(response.status).toBe(200);
    });

    it('Devrait créer un nouveau user avec les données fournies', async () => {
        jest.setTimeout(10000);
        const data = { nom: 'testUnitaire', prenom: 'testUnitaire', email : "testUnitaire", mdp :'testUnitaire' };

        const response = await request(app).post('/utilisateur').send(data);

        expect(response.statusCode).toBe(200);
        expect(response.body).toBeDefined();
     });

     it('Supprime le user créée', async () => {
        const response = await request(app).delete('/utilisateur/19');
        console.log(response);
        expect(response.status).toBe(200);
        expect(response.body).toBeDefined();
    });

    it('Devrait modifier user avec les données fournies 1', async () => {
        jest.setTimeout(20000);
        const data = { nom: 'test', prenom: 'test', email : "test", mdp :'test' };

        const response = await request(app).put('/utilisateur/1').send(data);

        expect(response.statusCode).toBe(200);
        expect(response.body).toBeDefined();
     });

     it('Devrait modifier user avec les données fournies 2', async () => {
        jest.setTimeout(10000);
        const data = { nom: 'Prenom1', prenom: 'Nom1', email : "Mail1", mdp :'Mdp1' };

        const response = await request(app).put('/utilisateur/1').send(data);

        expect(response.statusCode).toBe(200);
        expect(response.body).toBeDefined();
     });
});
```

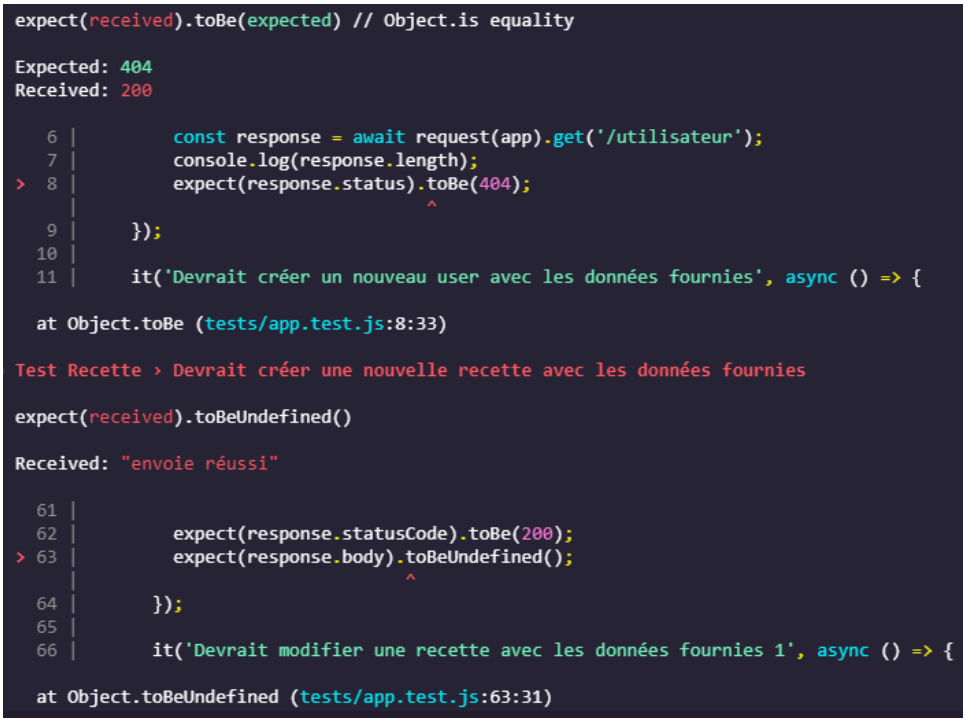
Ensuite les tests des méthodes de Recette :

```
describe("Test Recette", () => {
    it('/recette Devrait renvoyer le statut 200 OK', async () => {
        const response = await request(app).get('/recette');
        console.log(response.length);
        expect(response.status).toBe(200);
    });

    it('Devrait créer une nouvelle recette avec les données fournies', async () => {
        jest.setTimeout(10000);
        const data = { nom: 'testUnitaire', cuisson: 'testUnitaire', difficulte : "testUnitaire"};

        const response = await request(app).post('/recette').send(data);

        expect(response.statusCode).toBe(200);
        expect(response.body).toBeDefined();
      });

      it('Devrait modifier une recette avec les données fournies 1', async () => {
        const data = { nom: 'testUnitaire', cuisson: 'testUnitaire', difficulte : "testUnitaire"};
        const response = await request(app).put('/recette/1').send(data);

        expect(response.statusCode).toBe(200);
        expect(response.body).toBeDefined();
      });

      it('Devrait modifier une recette avec les données fournies 2', async () => {
        const data = { nom: 'nom1', cuisson: 'tps1', difficulte : "diff1"};
        const response = await request(app).put('/recette/1').send(data);

        expect(response.statusCode).toBe(200);
        expect(response.body).toBeDefined();
      });

      it('Supprime la recette créée', async () => {
        const response = await request(app).delete('/recette/6');
        console.log(response);
        expect(response.status).toBe(200);
        expect(response.body).toBeDefined();
    });
});
```

Pour les 2 classes les tests comprennent :

- Un test vérifiant la récupération de la classe (Methode Get)
- Un test vérifiant la création de la classe (Methode Add)
- Deux tests vérifiant la récupération de la classe (Methode Set)
- Un test vérifiant la supression de la classe (Methode Del)

Avant la finalisation de nos tests 2 erreurs était présente :



```
expect(received).toBe(expected) // Object.is equality

Expected: 404
Received: 200

   6 |         const response = await request(app).get('/utilisateur');
   7 |         console.log(response.length);
>  8 |         expect(response.status).toBe(404);
     |                                ^
   9 |     });
  10 |
  11 |     it('Devrait créer un nouveau user avec les données fournies', async () => {

  at Object.toBe (tests/app.test.js:8:33)

Test Recette › Devrait créer une nouvelle recette avec les données fournies

expect(received).toBeUndefined()

Received: "envoie réussi"

  61 |
  62 |         expect(response.statusCode).toBe(200);
> 63 |         expect(response.body).toBeUndefined();
     |                               ^
  64 |       });
  65 |
  66 |       it('Devrait modifier une recette avec les données fournies 1', async () => {

  at Object.toBeUndefined (tests/app.test.js:63:31)
```

La première était une erreur dans le code de réponse de la méthode GetUtilisateur. Le code attendu était 404 alors que le code reçu était 200. Cela était dû au fait que nous avions oublié de modifier ce code après avoir rajouté la méthode Get, précédemment absente.

La deuxième était une erreur dans l'attente du corps de la réponse de la méthode Set. Quelqu'un avait rajouté un message a la méthode alors que le test attendait une réponse vide.

Une fois résolu les tests devrait ressembler à cela :

```
 PASS  tests/app.test.js
  Test Utilisateur
    √ /utilisateur Devrait renvoyer le statut 200 OK (126 ms)
    √ Devrait créer un nouveau user avec les données fournies (1934 ms)
    √ Supprime le user créée (90 ms)
    √ Devrait modifier user avec les données fournies 1 (29 ms)
    √ Devrait modifier user avec les données fournies 2 (43 ms)
  Test Recette
    √ /recette Devrait renvoyer le statut 200 OK (15 ms)
    √ Devrait créer une nouvelle recette avec les données fournies (33 ms)
    √ Devrait modifier une recette avec les données fournies 1 (16 ms)
    √ Devrait modifier une recette avec les données fournies 2 (65 ms)
    √ Supprime la recette créée (163 ms)

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        3.865 s, estimated 4 s
Ran all test suites.
Jest did not exit one second after the test run has completed.
```