TP n°5 Les tests

Le but de ce TP est d'appliquer les bonnes pratiques en matière de tests [14]. Le code développé dans le TP4 n'étant pas le plus simple à tester, nous allons d'abord réécrire quelques bouts de code.

Remaniement du code du tp4

Créer le fichier `src/utils/readFile.js` exportant une fonction readFile transformant `fs.readFile` en promesse [6].

```
import fs from "fs"

export default function readFile(path) {
  // ...
}
```

Réécrivez `cardService.js` afin de correspondre au fichier ci-dessous.

```
import readFile from "../utils/readFile"

export function csvToJson(file) {
  // transforme un csv en json en utilisant l'en-tête du fichier pour définir les attr
}

export async function importBuildings() {
  const buildingsFile = await readFile(__dirname + "/../ressources/buildings.csv")
  return csvToJson(buildingsFile)
}

export async function importWorkers() {
  // ...
}
```

Ajout des tests unitaires

Commençons l'écriture des tests avec des tests unitaires. Dans un premier temps, il faut créer un fichier `src/services/cardService.test.js`.

⚠️ Les fichiers doivent se terminer par `.test.js` pour que Jest les retrouve automatiquement.

```
import * as cardService from "./cardService"

describe("csvToJson", () => {
  test("transform a csv to a javascript object", async () => {
    // ...
  })
  // ...
})
```

Vous devez écrire des tests unitaires uniquement pour `csvToJson` car `importBuildings` et `importWorkers` seront testés via des tests d'intégration.

ℹ️ l'accent grave `` ` `` permet d'écrire des chaînes de caractères sur plusieurs lignes (pratique pour écrire un faux csv).

ℹ️ la commande `npm run test` permet d'exécuter les tests.

Ajout des tests d'intégration

Il ne reste plus qu'à ajouter des tests d'intégration afin de vérifier que nos deux routes fonctionnent. Inspirez-vous de `src/routes/healthRouter.test.js` pour les écrire.

```
import request from "supertest"
import app from "../app"

describe("Test the health check", () => {
  test("It should response the GET method", async () => {
    const response = await request(app).get("/health")
    expect(response.statusCode).toBe(200)
    expect(response.body).toStrictEqual({ health: "ok" })
  })
})
```

La lecture d'un fichier étant un effet de bord, il est conseillé de *mocker* cette partie. L'exemple ci-dessous montre comment *mocker* la méthode *readFile* pour qu'elle nous retourne la chaîne de caractère `"foo"`.

```
import readFile from "../utils/readFile"
jest.mock("../utils/readFile")

readFile.mockImplementation((_path) => Promise.resolve("foo"))
```

⚠️ N'oubliez pas de tester les cas nominaux ainsi que les cas d'erreurs.

⚠️ Attention de vérifier que l'analyse de code statique (*ESLint*) ne révèle aucune erreur.

Maintenant que cette nouvelle fonctionnalité est testée, elle est enfin terminée 🎉 . Si vous étiez en équipe, ce serait le moment de créer la *merge-request* afin de faire valider votre code par vos coéquipiers afin de l'intégrer dans l'application.

Comme vous êtes seul, vous pouvez merger votre branche avec *git* (`git checkout master && git merge feature/cards-list`) ou via une *merge-request*.

Autres ressources

- Jest documentation [EN]
- Jest expect documentation [EN]
- Jest mock function documentation [EN]