Exécuter des requêtes SQL en asynchrone avec l'API *Promise*

Le module *mysql2* prend également en charge les promesses de résultat qui se couplent idéalement avec la syntaxe `async` et `await`. Pour mémoire, le mot-clé `async` s'utilise pour déclarer une fonction asynchrone et `await` permet, à l'intérieur d'une fonction asynchrone, d'attendre le résultat d'une autre fonction asynchrone.

L'objectif de ce TD consiste à modifier d'une part le service *db.js* et d'autre part les contrôleurs et les routes de l'application dorsale de gestion des missions afin de l'adapter à la syntaxe des *Promises*. Vous en profiterez également pour améliorer la sécurité de votre code SQL en utilisant les requêtes préparées.

Présentation du code de départ

Dans le dossier *debut*, vous trouverez l'ensemble des fichiers qui constituent l'application dorsale. Identifiez de suite les fichiers sur lesquels vous allez travailler :

- *services/db.js* : le script de connexion à la base de données ;
- *controllers/agents.js* : le contrôleur qui gère les requêtes concernant les agents ;
- *controllers/missions.js* : le contrôleur qui gère les requêtes concernant les missions ;
- *routes/agents.js* : le fichier de routage des requêtes concernant les agents ;
- *routes/missions.js* : le fichier de routage des requêtes concernant les missions.

Installation de l'application

Avant toute chose, déplacez-vous à la racine du répertoire *backend* et installez l'application :

```
cd backend
npm install
```

Le gestionnaire de packages *npm* se repose sur le fichier *package.json* afin d'installer la liste des dépendances dans un répertoire *node_modules*.

Il ne vous reste plus qu'à lancer l'application et à naviguer dessus :

```
npm start
```

Réécriture du service de connexion à la base de données

Tout d'abord, modifiez dans le fichier *services/db.js* la commande d'importation du module *mysql* de telle manière qu'il charge le support de l'API *Promise* :

```
const mysql = require('mysql2/promise');
```

Créez ensuite une fonction asynchrone nommée `query()` qui prenne deux paramètres :

- la requête SQL à exécuter ;
- la liste éventuelle des paramètres variables dans la requête SQL.

```
async function query(sql, params) {
    // instructions
};
```

Cette fonction sera chargée d'effectuer trois actions :

- attendre l'établissement d'une connexion à la base de données ;
- attendre l'exécution de la requête préparée à l'aide de la méthode `execute()` ;
- retourner la liste des résultats.

```
async function query(sql, params) {
    // create the connection
    const connection = await mysql.createConnection(config);
    // query database
    const [rows, fields] = await connection.execute(sql, params);

    return rows;
};
```

Il ne vous reste plus qu'à exporter la fonction :

```
module.exports = { query };
```

Réécriture des traitements concernant les agents

Étape 1 : définir un modèle

Créez un répertoire *models* à l'intérieur duquel vous consignerez dans un fichier *agents.js* les requêtes à la base de données en mode asynchrone. Dans ce fichier, commencez par importer le service de connexion à la base de données (il ne sera plus utile que dans ce fichier) :

```
const db = require('../services/db');
```

Écrivez ensuite une première fonction asynchrone `getAllAgents()` qui appelle la méthode `db.query()` pour lister tous les agents :

```
/* GET all agents */
async function getAllAgents() {

    const results = await db.query(
        `SELECT firstname, lastname, status, cap
        FROM agents, status
        WHERE ref_status = id_status;`
    );

    return results;
};
```

Écrivez une deuxième fonction `getOneAgent()` qui prenne en paramètre un `id_agent` afin de retourner les informations concernant un agent spécifique :

```
/* GET one agent */
async function getOneAgent(id_agent) {

    const results = await db.query(
        `SELECT firstname, lastname, status, cap
        FROM agents, status
        WHERE ref_status = id_status
        AND id_agent = ?;`,
        [id_agent]
    );

    return results;
};
```

Les deux dernières fonctions à écrire servent à récupérer pour la première toutes les missions d'un agent et, pour la seconde, les informations d'une mission spécifique :

```
/* GET all missions of an agent */
async function getAllMissionsAgent(id_agent) {

    const results = await db.query(
        `SELECT country, cost, date_from, date_to
        FROM missions
        WHERE ref_agent = ?;`,
        [id_agent]
    );

    return results;

};

/* GET a mission of a specific agent */
async function getOneMissionAgent(id_agent, id_mission) {

    const results = await db.query(
        `SELECT firstname, lastname, status, cap, country, cost, date_from, date_to
        FROM agents, missions, status
        WHERE ref_agent = ?
        AND id_mission = ?
        AND id_agent = ref_agent
        AND ref_status = id_status;`,
        [id_agent, id_mission]
    );

    return results;

}
```

N'oubliez pas d'exporter toutes ces fonctions :

```
module.exports = {
    getAllAgents,
    getOneAgent,
    getAllMissionsAgent,
    getOneMissionAgent
};
```

Étape 2 : utiliser le modèle dans le contrôleur

Le contrôleur va maintenant subir un toilettage en profondeur. Appelez le seul module utile, celui du modèle *models/agents.js* :

```
const agents = require('../models/agents');
```

Définissez à présent des méthodes asynchrones pour récupérer :

- la liste de tous les agents ;
- les informations à propos d'un agent ;
- les informations à propos d'une mission spécifique effectuée par un agent particulier.

```
const getAgents = async (req, res, next) => {
    try {
        res.send(await agents.getAllAgents());
    } catch (err) {
        next(err);
    }
};

const getAgent = async (req, res, next) => {
    try {
        res.send(await agents.getOneAgent(req.params.id_agent));
    } catch (err) {
        next(err);
    }
};

const getMissionAgent = async (req, res, next) => {
    try {
        res.send(await agents.getOneMissionAgent(req.params.id_agent,
req.params.id_mission));
    } catch (err) {
        next(err);
    }
};
```

Il reste une dernière méthode à définir, celle qui liste toutes les missions d'un agent et qui repose sur deux fonctions importées dans le modèle :

- `getOneAgent()` pour récupérer les informations d'un agent ;
- `getAllMissionsAgent()` pour récupérer toutes les missions d'un agent.

```
const getMissionsAgent = async (req, res, next) => {
    try {
        // which agent?
        const agent = await agents.getOneAgent(req.params.id_agent);
        // which missions?
        const missions = await agents.getAllMissionsAgent(req.params.id_agent);

        res.send({
            'firstname': agent[0]['firstname'],
            'lastname': agent[0]['lastname'],
            'status': agent[0]['status'],
            'cap': agent[0]['cap'],
            'missions': missions
        });
    } catch (err) {
        next(err);
    }
};
```

En dernier lieu, exportez toutes ces méthodes :

```
module.exports = {
    getAgents,
    getAgent,
    getMissionsAgent,
    getMissionAgent
};
```

Étape 3 : reconfigurer les routes

Le fichier *routes/agents.js* devient encore plus clair qu'auparavant grâce à tous ces ajustements.

Vous définirez à l'intérieur quatre routes :

- `/` qui fera appel à la méthode `agents.getAgents` ;
- `/:id_agent` qui fera appel à la méthode `agents.getAgent` ;
- `/:id_agent/missions` qui fera appel à la méthode `agents.getMissionsAgent` ;
- `/:id_agent/missions/:id_mission` qui fera appel à la méthode `agents.getMissionAgent` ;

À vous de jouer !

Suivez à nouveau toutes ces étapes pour les opérations concernant le traitement des missions.

Vous trouverez le code final des documents HTML et JavaScript dans le dossier *fin* de ce second TD.