Sécuriser une application dorsale

La sécurité d'une application chargée de gérer des données sensibles comme les nom et prénom des agents d'une unité est fondamentale. La décision prise lors du premier TD sur l'authentification de ne pas assujettir les routes *GET* au contrôle d'accès est fortement discutable. Elle sera réévaluée au cours de ce TD pour offrir au final une application complètement sécurisée.

D'un point de vue pratique, vous allez mettre en place un mécanisme pour ajouter des utilisateurs en chiffrant leurs mots de passe à l'aide du module *Bcrypt* et pour leur permettre de s'authentifier à l'aide de leurs identifiants (email et mot de passe) afin de naviguer à travers les routes de l'application. Le processus d'authentification sera complexifié grâce au recours au module *JSONWebToken* chargé de générer un jeton (*token*) à partir d'une phrase secrète si les identifiants renseignés par l'utilisateur sont cohérents.

Étape 1 : lister les utilisateurs

Compléter le modèle des utilisateurs

Complétez le module *models/users.js* avec des méthodes pour :

- lister tous les utilisateurs ;
- obtenir le détail d'un utilisateur par son email.

```
/* GET all users */
async function getAllUsers() {

    const results = await db.query(
        `SELECT email, password
        FROM users
        ORDER BY email ASC;`
    );

    return results;
};

/* GET user by email */
async function getUserByEmail(email) {

    const results = await db.query(
        `SELECT id_user, password
        FROM users
        WHERE email = ?;`,
        [email]
    );

    return results[0];
};
```

Et n'oubliez pas de les exporter via `module.exports` !

Un contrôleur pour lister les utilisateurs

Créez un fichier *users.js* dans le répertoire *controllers* et définissez deux fonctions asynchrones pour d'une part lister tous les utilisateurs et d'autre part obtenir les informations d'un seul d'entre eux :

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

const getUsers = async (req, res, next) => {
    try {
        res.send(await users.getAllUsers());
    } catch (err) {
        next(err);
    }
};

const getUser = async (req, res, next) => {
    try {
        res.send(await users.getUserById(req.params.id_user));
    } catch (err) {
        next(err);
    }
};

module.exports = {
    getUsers,
    getUser
};
```

Tracer les routes

Créez à présent un fichier *users.js* dans le répertoire *routes* et tracez les routes *ad hoc* pour lister les utilisateurs :

```
const express = require('express');
const router = express.Router();
const users = require('../controllers/users');

router.get('/', users.getUsers);
router.get('/:id_user', users.getUser);

module.exports = router;
```

Il ne vous reste plus qu'à importer ce routeur dans le fichier principal *app.js* et à l'utiliser :

```
var usersRouter = require('./routes/users');

app.use('/users', usersRouter);
```

Étape 2 : ajouter un utilisateur

Dansun premier temps, écrivez dans le modèle *users.js* une méthode asynchrone `addOneUser()` pour insérer un utilisateur dans la base de données qui accepte deux paramètres : un email et un mot de passe. Exportez cette méthode à la fin du module.

Dans un second temps, écrivez un contrôleur `addUser()` qui fasse appel à cette méthode et transmettez-lui les paramètres présents dans l'objet `req.body` en prenant soin de chiffrer préalablement le mot de passe avec la méthode `hash()` du module *Bcrypt* :

```
const password = await bcrypt.hash(req.body.password, 10);
```

Comme ce contrôleur utilise une librairie externe, vous devez d'abord l'installer avec *npm* :

```
npm install bcrypt --save
```

Et l'importer dans le fichier :

```
const bcrypt = require('bcrypt');
```

Enfin, définissez une route *POST* `/add` pour l'insertion d'un utilisateur.

Étape 3 : authentifier un utilisateur

Étape 4 : appliquer le contrôle d'accès

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