TD 4 SR10/AI16

Récapitulatif

Dans la séance précédente, nous nous sommes concentrés sur la récupération des données depuis votre base de données et leur affichage sur différentes vues. Pour cette séance, nous allons nous pencher sur l'ajout et la modification de nouvelles données dans votre base de données, en utilisant des formulaires, tout en abordant également la gestion des sessions pour sécuriser votre application.

Partie 4 Enregistrement des données

1. Utiliser votre formulaire de création d'un nouveau compte pour un utilisateur.
    a. Votre formulaire doit avoir la valeur « post » pour l'attribut method, et la valeur d'une route dédiée (par exemple, /users/users) au traitement de ce formulaire pour l'attribut action.
    b. Compléter le code ci-dessous dans le fichier routes/user.js :

```
//initialisation
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
……
router.post('/users', function (req, res, next) {

/*récupérer les données passées via le body de la requête post :
Exemple :
   const user_fname = req.body.fname;
   const user_lname = req.body.lname;

*/
…
//utiliser le model pour enregistrer les données récupérées dans la BD
……

  });
```

2. Traiter de la même manière les autres formulaires de création de votre application :
    a. Ajout d'une organisation.
    b. Ajout d'une fiche de poste.
    c. Ajout d'une offre.
    d. Ajout d'une candidature. Cette fonctionnalité nécessite d'implémenter le code qui permet de charger des fichiers sur le serveur nodejs. Nous abordons ce point à la fin du TD.

Partie 5 Connexion d'un utilisateur et gestion des sessions

Vous avez deux possibilités pour gérer l'authentification dans une application express :

1. Authentification basique : utiliser simplement le middlware « express-sessions ».
2. Authentification via le middleware « passeport » qui gère l'authentification d'une manière avancée avec plusieurs options de configuration possible.

Dans ce TD, on privilège la première option.

Dans ce cas, il faut suivre les étapes suivantes :

- Etape 1 : installer et configurer les modules nécessaires.

```
npm install express-session
```

```
const express = require('express');
const session = require('express-session');

const app = express();

app.use(session({
  secret: 'votre_secret',
  resave: false,
  saveUninitialized: false,
  // Autres options comme cookie, durée de vie, etc.
}));
```

- Etape 2 : élaborer le contrôleur chargé du traitement du formulaire d'authentification, et qui autorise la création d'une session utilisateur une fois les informations validées.

```
app.post('/login', (req, res) => {
  // Vérification des informations d'identification de l'utilisateur
  if (req.body.username === 'user' && req.body.password === pwd) {
    // Création d'une session utilisateur
    req.session.user = req.body.username;
    // Ajouter le rôle aussi dans la session
    Req.session.role = 'user' ;
    res.send('Authentification réussie !');
  } else {
    res.send('Nom d\'utilisateur ou mot de passe incorrect.');
  }
});
```

- Etape 3 : identifier clairement les routes sécurisées et en spécifiant les rôles requis pour y accéder pour optimiser l'authentification en exploitant la session pour vérifier continuellement si l'utilisateur est authentifié à chaque accès à une route nécessitant une authentification. Si l'utilisateur n'est pas authentifié, le rediriger automatiquement vers la page de connexion. En outre, il est essentiel de vérifier le rôle de l'utilisateur et s'il a les autorisations nécessaires pour effectuer l'action souhaitée. Dans le cas contraire, afficher un message d'erreur adapté pour informer l'utilisateur de l'indisponibilité de cette fonctionnalité avec son rôle.

```
app.get('/profil', (req, res) => {
  if (req.session.user) {
    res.send('Bienvenue sur votre profil, ' + req.session.user + '!');
  } else {
    res.redirect('/login');
```

```
        }
});
```

- Etape 4 : traiter l'action déconnexion.

```
app.get('/logout', (req, res) => {
        req.session.destroy((err) => {
          if (err) {
            console.log(err);
          } else {
            res.redirect('/login');
          }
        });
});
```

Exemple d'implémentation d'une authentification

Dans cet exemple, j'implémente deux types de rôle admin et user.

Le fichier « session.js » abrite le code permettant l'initialisation de la session ainsi que les opérations associées à celle-ci :

```
var sessions = require("express-session");
module.exports = {
  init: () => {
    return sessions({
      secret: "xxxzzzyyyaaabbbcc",
      saveUninitialized: true,
      cookie: { maxAge: 3600 * 1000 }, // 60 minutes
      resave: false,
    });
  },
  creatSession: function (session, mail, role) {
    session.userid = mail;
    session.role = role;
    session.save(function (err) {
      console.log(err);
    });
    return session;
  },

  isConnected: (session, role) => {

    if (!session.userid || session.userid === undefined) return false;
    if (role && session.role !== role) return false;
    return true;
  },
```

```
  deleteSession: function (session) {
    session.destroy();
  },
};
```

Intégrer le fichier « session.js » dans le fichier app.js et centraliser la protection des routes de manière plus efficace à travers ce dernier :

```
…
var session=require('./session');
…
// check user before app.use (path, router)
app.all("*", function (req, res, next) {
  const nonSecurePaths = ["/signin", "/signup"];
  const adminPaths = []; //list des urls admin
  if (nonSecurePaths.includes(req.path)) return next();

  //authenticate user
  if (adminPaths.includes(req.path)) {
    if (session.isConnected(req.session, "admin")) return next();
    else
      res
        .status(403)
        .render("error", { message: " Unauthorized access", error: {} });
  } else {
    if (session.isConnected(req.session)) return next();
    // not authenticated
    else res.redirect("/signin");
  }
});

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