TD 3 SR10/AI16

Sur deux séances

Partie 1 – préparation de l'environnement de développement

L'objectif de cette partie est la création d'un premier projet express (framework node.js). Pour le faire, il faut suivre les étapes suivantes :

- Installer le package **express-generator** (via le terminal)

>npm install -g express-generator
>express -h

- Créer un projet express (EJS est choisi comme moteur de visualisation) :

  > express --view=ejs myapp

```
create : myapp\
create : myapp\public\
create : myapp\public\javascripts\
create : myapp\public\images\
create : myapp\public\stylesheets\
create : myapp\public\stylesheets\style.css
create : myapp\routes\
create : myapp\routes\index.js
create : myapp\routes\users.js
create : myapp\views\
create : myapp\views\error.ejs
create : myapp\views\index.ejs
create : myapp\app.js
create : myapp\package.json
create : myapp\bin\
create : myapp\bin\www
```

Description de la commande express : génère un dossier qui a le même nom que le projet. Dans ce dossier, plusieurs sous dossiers et fichiers sont générés automatiquement. Le tableau ci-dessous décrit les différents éléments générés par express.

Dossier/fichier	Description
public	Les ressources statiques (html, css, javascript, images)
routes	Contient les scripts JS qui traitent les requêtes http
routes\index.js	Traiter les requêtes de l'url home (/)
routes\users.js	Traiter les requêtes de l'url (/users)
view\	Contients les templates ejs
view\error.ejs	Template (visuel) de la page d'erreur
view\index.ejs	Template (visuel) de la page accueil
app.js	Le point d'entrée d'une application express
package.js	Les dépendances
bin\www	Configurer et lancer le serveur http

- Les commandes pour tester cette application :
  1. Aller dans le dossier de l'application :

> cd myapp

2. Installer les dépendances :
   > npm install

3. Lancer l'application (windows):
   > SET DEBUG=myapp:* & npm start

Questions :

- Analyser les fichiers et les dossiers crées par express-generator

Parie 2 – Création de la partie modèle

Dans cette partie, On ajoute un dossier « model » qui représente la partie modèle de l'architecture MVC de votre projet. Dans ce dossier, on commence d'abord par créer un script js pour la connexion à la base de données MySQL :

```
var mysql = require("mysql");

var pool = mysql.createPool({
    host: "tuxa.sme.utc", //ou localhost
    user: "ai16pxxxx",
    password: "**********",
    database: "ai16pxxx"
});

module.exports = pool;
```

Par la suite, on doit créer pour chaque entité de notre modèle un fichier « js » qui regroupe les opérations de persistance CRUD (create, read, update, delete). Si on prend par exemple la table utilisateur (version simplifiée pour l'exemple) :

```
CREATE TABLE `Utilisateur` (
  `email` varchar(25) NOT NULL,
  `pwd` varchar(25) NOT NULL,
  `nom` varchar(25) NOT NULL,
  `prenom` varchar(25) NOT NULL,
  `type` varchar(10) NOT NULL
)
```

Un exemple de modèle pour cette table :

```
var db = require('./db.js');
module.exports = {
    read: function (email, callback) {

        db.query("select * from Utilisateur where email= ?",email, function
(err, results) {
            if (err) throw err;
            callback(results);

        });

    },
    readall: function (callback) {
        db.query("select * from Utilisateur", function (err, results) {
            if (err) throw err;
            callback(results);

        });

    },
    areValid: function (email, password, callback) {
        sql = "SELECT pwd FROM USERS WHERE email = ?";
        rows = db.query(sql, email, function (err, results) {
            if (err) throw err;
            if (rows.length == 1 && rows[0].pwd === password) {
                callback(true)
            } else {
                callback(false);
            }
        });



    },

    creat: function (email, nom, prenom, pwd, type, callback) {
        //todo
        return false;

    }
}
```

Question :

- Développer l'ensemble des scripts du modèle de votre projet (utilisateur, offre, candidature, etc.).

Partie 3 – connecter le modèle, la vue et le contrôleur

Dans cette partie, nous allons créer des routes (intègrent des contrôleurs) qui vont traiter les requêtes http (de type get) et renvoyer une vue.

- L'ajout d'une route dans le contrôleur « routes/user.js ». Cette route a comme url : http://localhost:3000/users/userslist et elle retourne une vue qui affiche la liste de tous les utilisateurs dans un tableau (Array js).

```
……
router.get('/userslist', function (req, res, next) {

  result=userModel.readall(function(result){

    res.render('usersList', { title: 'List des utilisateurs', users:
result });

  });
…….
```

- L'ajout d'une vue « userListe » (userList.ejs) :

```
<body>
    <h1>
        <%= title %>
    </h1>
    <table border>
        <thead>
            <tr>
                <th>Firstname</th>
                <th>Lastname</th>
                <th>Email</th>
            </tr>
        </thead>
        <tbody>
            <% users.forEach((user)=> { %>
                <tr>
                    <td>
                        <%= user.prenom %>
                    </td>
                    <td>
                        <%= user.nom %>
                    </td>
                    <td>
                        <%= user.email %>
                    </td>
                </tr>
                <% }) %>
```

```
        </tbody>

    </table>

</body>
```

Questions :

- Adapter et ajouter des routes pour traiter les requêtes http (get) de votre application.
- Utiliser vos templates html/css/bootstrap pour créer les vues EJS de votre projet qui permettent de visualiser les données de votre base de données (liste des utilisateurs, liste des offres, liste des organisations, etc.).

Ressources

- https://github.com/mysqljs/mysql#pooling-connections
- https://ejs.co/