Node, Express, MongoDB

- **Nous allons ajouter la mise à jour dans notre Express App.**
  - Pour ce faire nous allons créer une nouvelle collection dans **MongoDB**, via **Mongo Mangement Studio** ou le **shell** pour stocker des utilisateurs : “**users**”. On créé 1 user via le shell MongoDB (commande mongo) :

```
{
    "name": "MASCARON",
    "firstName": "Stéphane",
    "login": "steph",
    "mdp": "azerty",
    "function": "Free Software Architecte",
    "office": "SMaLL Stéphane MASCARON Architecte Logiciels Libres",
    "date_naiss": "Wed Apr 22 1970 08:30:22 GMT+0100 (CET)",
    "adresse1": "xxx avenue des freesoftwares",
    "adresse2": "",
    "cp": "40000",
    "city": "MONT-DE-MARSAN",
    "mobile_phone": "0609090909”,
    "home_phone": "0558080808"
}
```

  - Créez un contrôleur « **routes/users.js** », une vue « **views/users.hbs** » qui affiche les données de l’utilisateur. Pensez à charger le module dans app.js et à définir la route (use).

Node, Express, MongoDB

- **Insérer l'utilisateur via le Shell MongoDB :**
  - Lancer un shell mongo :

```
$ mongo
```

```
>use gretajs
switched to db gretajs
> db.createCollection('users')
{ "ok" : 1 }
> db.users.insert({name:"MASCARON", firstName: "Stephane", login: "steph", mdp: "azerty",
function: "Free Software Architect", office: "EI SMALL", date_naiss: "22/04/29170",
adresse1: "xxxx rue des logiciels libres", adresse2: "", cp: "40000", city: "MONT-DE-MARSAN",
mobile_phone: "0606060606", home_phone: "0558080808"})

WriteResult({ "nInserted" : 1 })
> db.users.find()
{ "_id" : ObjectId("5babd87c5a4a7fdc4b7e3a9f"), "name" : "MASCARON",
"firstName" : "Stephane", "login" : "steph","mdp" : "azerty",
"function" : "Free Software Architect", "office" : "EI SMALL",
"date_naiss" : "22/04/29170", "adresse1" : "xxxx rue des logiciels libres",
"adresse2" : "", "cp" : "40000", "city" : "MONT-DE-MARAN",
"mobile_phone" : "0606060606", "home_phone" : "0558080808" }
```

  - De la même façon créez votre utilisateur avec votre nom.

Nb : cette collection « users" va nous permettre plus tard de gérer l'authentification dans l'application (login, mdp).

Node, Express, MongoDB

- Nous devons créer **un contrôleur**, **une vue** et **une action** qui affiche la liste des users : "**/users**" dont le contrôleur sera "**users.js**" :

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
  global.db.collection('users').find().toArray(function(err, result) {
    if (err) {
      throw err;
    }
    console.log('users: ', result);
    res.render('users', {title: 'List of users', users: result});
  });
});

module.exports = router;
```

NB: Dans ce contrôleur nous réalisons une requête sur la collection "users" pour laquelle on récupère l'ensemble des utilisateurs (db.collection('users').find() ...)

Node, Express, MongoDB

- Il faut créer la vue pour cette liste des utilisateurs, on va utiliser un select pour afficher les utilisateurs : “**users.hbs**”

```
<h1> Express App : {{title}}</h1>
<br>
<select name="users" id="user">
    {{#compare users null operator="=="}}
        <option value="rien">Il n'y a pas de données dans la base users</option>
    {{else}}
           <option value="0" >Select a user to modify his datas</option>
        {{#each users}}
            <option value="{{this._id}}" >{{this.name}} {{this.firstName}}-
                                                    {{this.function}}</option>
        {{/each}}
    {{/compare}}
</select>
<h2>or</h2><br/>
<button id="newUser">Create a new user</button>
<script>
    var usr = document.getElementById('user');
    usr.addEventListener('change', function(evt) {
       window.location = "/formUser/"+usr.value;
    });
    var btn_new = document.getElementById('newUser');
    btn_new.addEventListener('click', function(evt) {
       window.location = "/newUser";
    });
</script>
```

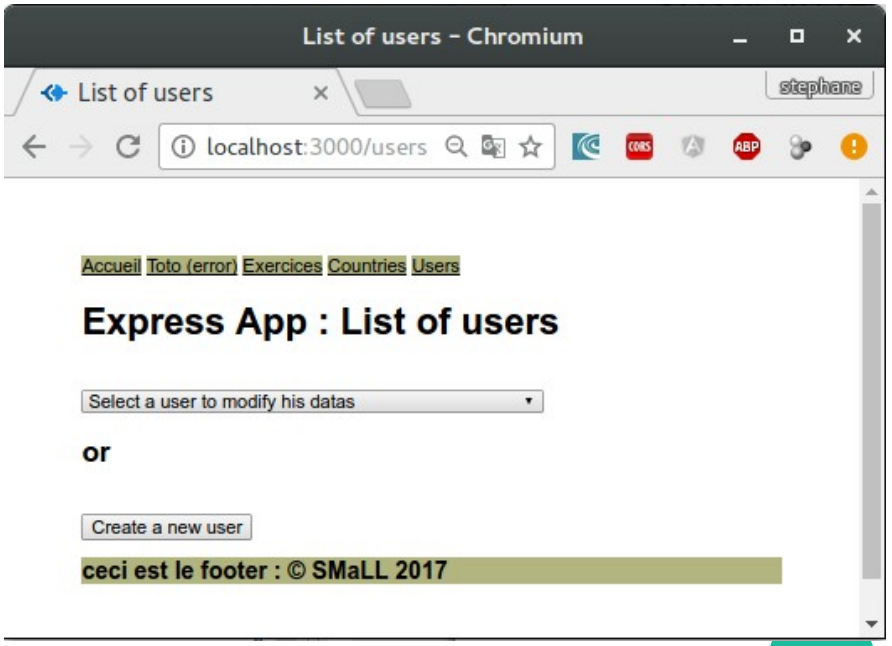
Node, Express, MongoDB

- Nous allons ajouter dans notre partials “**nav.hbs**” le lien permettant d’accéder à cette liste des users :

```
<nav class="barnav">
    <ul id="nav">
        <li><a href="/">Accueil</a></li>
        <li><a href="/toto">Toto (error)</a></li>
        <li><a href="/exos">Exercices</a></li>
        <li><a href="/countries">Countries</a></li>
        <li><a href="/users">Users</a></li>
    </ul>
</nav>
```

- Voici le résultat :

Si vous sélectionnez un user dans la liste le script déclenchera automatiquement un appel à l’action /formUser qui affiche un formulaire avec les données prêtes à être modifiées. Le bouton “Create …” affiche la vue formUser.hbs vide de données prête pour une saisie d’un nouvel utilisateur.



Node, Express, MongoDB

- **Mettons en oeuvre cette mise à jour dans notre Express App.**
  - Il faut initialiser les routes pour chaque action (pathname) dans **app.js** à deux endroits :

```
(…)
var formUser    = require('./routes/formUser');
var modifyUser  = require('./routes/modifyUser');
(…)
```

  - Puis nous allons gérer les routes, qui seront plus complexes que pour une sélection globale :

```
(…)
app.use('/formUser', formUser);  // affichera le formulaire
app.use('/modifyUser', modifyUser);  // Enregistre les données dans la base
(…)
```

Node, Express, MongoDB

- **Vous allez devoir créer 2 vues et 2 contrôleurs :**

1- formUser.hbs :

```
<h1> Express App : {{title}}</h1>
<form methode="POST" action="{{form_action}}" id="formAddUser">
   <input type="hidden" name="id" value="{{user._id}}" />
    name : <input type="text" name="name" value="{{user.name}}" /><br/>
    firstName : <input type="text" name="firstName" value="{{user.firstName}}" /><br/>
    login : <input type="text" name="login" value="{{user.login}}" /><br/>
    mdp : <input type="password" name="mdp" value="{{user.mdp}}" /><br/>
    function : <input type="text" name="function" value="{{user.function}}" /><br/>
    office : <input type="text" name="office" value="{{user.office}}" /><br/>
    date_naiss : <input type="text" name="date_naiss" value="{{user.date_naiss}}" /><br/>
    adresse1 : <input type="text" name="adresse1" value="{{user.adresse1}}" /><br/>
    adresse2 : <input type="text" name="adresse2" value="{{user.adresse2}}" /><br/>
    cp : <input type="text" name="cp" value="{{user.cp}}" /><br/>
    city : <input type="text" name="city" value="{{user.city}}" /><br/>
    mobile_phone : <input type="text" name="mobile_phone" value="{{user.mobile_phone}}" /><br/>
    home_phone : <input type="text" name="home_phone" value="{{user.home_phone}}" /><br/>
    <input type="submit" value="Confirmer la {{libelle}}" />
</form>

<!-- Script de validation du formulaire de modification -->
<script>
    var form = document.getElementById('formAddUser');
    form.addEventListener('submit', function(evt) {
        form.action += "/{{user._id}}";
        form.submit();
    });
</script>
```

2- modifyUser.hbs :

```
<h1> Express App : {{title}}</h1>
<ul>
    <li>name :.......................... {{user.name}}</li><br/>
    <li>firstName :..................... {{user.firstName}}</li><br/>
    <li>login :......................... {{user.login}}</li><br/>
    <li>mdp :........................... {{user.mdp}}</li><br/>
    <li>function :...................... {{user.function}}</li><br/>
    <li>office :........................ {{user.office}}</li><br/>
    <li>date_naiss :.................... {{user.date_naiss}}</li><br/>
    <li>adresse1 :...................... {{user.adresse1}}</li><br/>
    <li>adresse2 :...................... {{user.adresse2}}</li><br/>
    <li>cp :............................ {{user.cp}}</li><br/>
    <li>mobile_phone :.................. {{user.mobile_phone}}</li><br/>
    <li>home_phone :.................... {{user.home_phone}}</li><br/>
</ul>
```

- La première vue affiche un formulaire, qui sera utilisé à la fois pour les modifications d'un utilisateur existant ou pour la création. La seconde vue permet d'afficher les données qui viennent d'être modifiées dans la base de données MongoDB.

3- formUser.js :

```
var express = require('express');
var router = express.Router();
var ObjectID = require('mongodb').ObjectID;

/* GET users from _id. */
router.route('/:_id').get(function(req, res) {
 console.log('req.originalUrl : ' , req.originalUrl);
  global.db.collection('users')
     .find({_id: new ObjectID(req.params._id)})
     .toArray(function(err, result) {
    if (err) { throw err; }
    console.log('formUser: ', result);
    res.render('formUser', {
        title: "Form user\'s datas",
        libelle: "modification",
        form_action: "/modifyUser",
        user: result[0] // il n'y a qu'une réponse possible puisque requête via _id user
    });

  });
});

module.exports = router;
```

NB : On remarque dans le paramètre de la fonction route(‘/:_id’), il permet de filtrer sur une valeur dans l’URL c’est le principe du protocole REST très utilisé pour développer des APIs onlines.

4- modifyUser.js : 3 Blocs imbriqués, 3 Callbacks !

```
var express = require('express');
var router = express.Router();
var ObjectID = require('mongodb').ObjectID;
/* SET user from _id with new data for an update into mongoDB . */
router.route('/:_id').get(function (req, res) {
    console.log('req.originalUrl : ', req.originalUrl);
    global.db.collection('users').update(
     {_id: new ObjectID(req.params._id)},
     {$set: req.query},
       function (err, result) {
         if (err) { throw err; }
         console.log('modifyUser: ', result);
         global.db.collection('users').find({ _id: new new ObjectID(req.params._id)
         }).toArray(function (err, result) {
              if (err) { throw err; }
              console.log('users: ', result);
              res.render('modifyUser', {
                   title: 'User modified without error',
                   user: result[0]
              });
          }); // fin du find() après update
       } // fin callback de l'update
    ); // fin de l'update()
}); // fin de la gestion de la route
module.exports = router;
```

NB : On remarque le même filtre par l'_id dans l'URL, mais on enchaîne ici un select après l'update sur l'_id pour ré-afficher les données modifiée dans la base.

- **Il ne nous reste plus qu'à définir 2 actions pour la création d'un utilisateur :**
  - Une pour afficher le formulaire de saisie : "**/newUser**" et l'autre pour l'insertion effective dans la base de données, suite à la validation du formulaire : "**/createUser**" dans **app.js** :

```
(…)
var newUser     = require('./routes/newUser');
var createUser  = require('./routes/createUser');
(…)
```

  - Cela doit devenir pour vous un reflex, il faut donc deux contrôleurs et une seule vue, car on va réutiliser la vue **formUser.hbs** pour la création d'un utilisateur.

- **newUser.js** : le contrôleur qui appelle le formulaire de saisie **formUser.hbs** :

```
var express = require('express');
var router = express.Router();

/* GET formUser page to insert a new user */
router.get('/', function(req, res, next) {
    res.render('formUser',
        {title: 'Create a new user',
         libelle: "creation",
         form_action: "/createUser"
    });
});
module.exports = router;
```

NB : Vous pouvez remarquer que la fonction **“render”** qui appelle **“formUser”** reçoit également **en paramètre** un objet littéral qui contient **le titre de la page**, le **libellé du bouton** de validation du formulaire et **l’action du formulaire**, on pourrait paramétrer également la méthode (POST/GET) etc …

Cela nous permet de réutiliser la vue modification pour la saisie.

Node, Express, MongoDB

- **createUser.js** : est le contrôleur qui va exécuter la fonction d'insertion dans la base de données :

```
var express = require('express');
var router = express.Router();
/* Insert one new user into database. */
router.route('/').get(function (req, res) {
    console.log('req.originalUrl : ', req.originalUrl);
    global.db.collection('users').insert([req.query],
        function (err, result) {
            if (err) {
                throw err;
            }
            console.log('createUser: ', result);
            res.render('modifyUser', {
                title: 'Creating User without error with datas below :',
                user: result.ops[0]
            });
        } // fin callback de l'insert
    ); // fin de l'insert()
}); // fin de la gestion de la route
module.exports = router;
```

NB : On remarque la syntaxe de la fonction "**insert**" qui prend comme paramètre un tableau d'objet. Nous lui passons req.query qui contient les données du formulaire au format JSON. Regardez en debug (via node-inspector) la variable **result** ...

Node, Express, MongoDB

- **Conclusion :**
  - Vous avez vu comment à partir **d'Express**, **MongoDB** et **Handlebars** construire une application Web simple mais qui repose sur une **architecture MVC** qui a fait ses preuves.
  - On remaquera que pour d'importantes applications Web, le fait de devoir **ajouter dans app.js** l'ensemble **des routes et des actions** peut vite devenir **difficilement maintenable**.
  - Il serait intéressant de **réfléchir** à une **organisation de la gestion des routes** (ou actions ex. : **"/users"**) plus **générique**.