Node, Express, MongoDB

- **Écriture d'un module générique pour lire des données : /routes/finder.js**
  - Illustration avec le contrôleur « finder.js » permettant de requêter la collection passée en paramètre dans le fichier config_actions.json dans la variable : **model**

```
/* ******************************************************
**  Module générique pour faire un "find()" sans filtre *
** ******************************************************/
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');

/* GET users listing. */
router.get('/', function (req, res, next) {
    var path = "/" + req.originalUrl.split('/')[1];
    var type = req.method;
    global.schemas[global.actions_json[type + path].model].find({}, function (err, result) {
        if (err) { throw err;}
        console.log(result);
        if (result.length == 0) result = null;
        res.render(global.actions_json[type + path].view,
                   { title: 'List of results :', data: result }
        );
    });
});
module.exports = router;
```

Node, Express, MongoDB

- **Écriture d'un module générique pour supprimer des données : /routes/delete.js**
  - Illustration avec le contrôleur «delete.js » permettant de supprimer de la collection un enregistrement via son _id :

```
/* ********************************************************
**  Module générique pour faire un "deleteOne()" via l'_id *
** ********************************************************/
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');
var ObjectId = mongoose.Types.ObjectId;

/* DELETE record from _id into url and into config_actions.json */
router.route('/:_id').get(function (req, res) {
    var path = "/" + req.originalUrl.split('/')[1];
    var type = req.method;
    var model = global.actions_json[type + path].model;
    global.schemas[model].deleteOne({_id: new ObjectId(req.params._id)}, function (err, result) {
        if (err) {
            throw err;
        }
        global.schemas[model].find({}, function (err, result2) {
            console.log("result after delete : ", result2);
            res.render(global.actions_json[type + path].view, {
                title: "List of " + model,
                data: result2
            });
        });
    });
});
module.exports = router;
```

Node, Express, MongoDB

- **Écriture d'un module générique pour lire et liéer des données entre 2 collections « populate » :**
  - Illustration avec le contrôleur «**populate.js** » permettant de lire une collection et de récupérer par une clé externe les données d'un autre collection via l'id par exemple :

```
/* ********************************************************
**  Module générique pour faire un "populate()" via l'_id *
** ********************************************************/
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');

/* GET users listing. */
router.get('/', function (req, res, next) {
    var path = "/" + req.originalUrl.split('/')[1];
    var type = req.method;
    var model = global.actions_json[type + path].model;
    var pop_ref = global.actions_json[type + path].pop_ref;
    global.schemas[model].find({}).populate(pop_ref).exec(function (err, result) {
        if (err) {
            console.log("error: ", err);
            return handleError(err);
        } else {
            if (result.length == 0) result = null;
            res.render(global.actions_json[type + path].view, { title: 'List of results', data: result });
        }
    });
});

module.exports = router;
```

Node, Express, MongoDB

- **Écriture de la configuration dans le fichier « config_actions.json » pour ces 3 contrôleurs :**
  - Illustration avec les action suivantes en lien avec le schéma de la collection :

```
(...)
   "GET/users": {
        "controler": "finder",
        "view": "users",
        "model": "Users"
    },
     "GET/deleteUser": {
        "controler": "delete",
        "view": "users",
        "model": "Users"
    },
    "GET/userspopulate" :  {
        "controler": "populate",
        "view": "userspopulate",
        "model": "Users",
        "pop_ref": "countries"
    },
(...)
```

```
16          }
17      },
18      "Users": {
19          "collection": "users",
20          "schema": {
21              "_id": "ObjectId",
22              "name": "String",
23              "firstName": "String",
24              "login": "String",
25              "mdp": "String",
26              "function": "String",
27              "office": "String",
28              "date_naiss": "String",
29              "adresse1": "String",
30              "adresse2": "String",
31              "cp": "String",
32              "city": "String",
33              "countries": {
34                  "type": "String",
35                  "ref": "Countries"
36              },
37              "mobile_phone": "String",
38              "home_phone": "String"
39          }
40      }
41  }
```

- **Exercice T P** : Intégrez ces éléments dans votre applications et créez le contrôleur qu'il manque : « **update.js** » qui permet de modifier un enregistrement d'une collection.

  NB : Penser à changer dans les **{{#each }}** des vues le nom de la variables qui contient les données par la variable « **data** » puisque dans les render() nous devons être générique sur les termes nous renvoyons : « **{data: result}** » .