- **Mongoose : Un outil pour “schématiser” l’accès à la base de données**
  - Mongoose est une librairie qui permet de créer des abstractions objets sur les bases de données MongoDB. (équivalent de hibernate en Java)
  - Pour l’installer, il faut ajouter mongoose :

```
$ npm install mongoose --save
```

  - Dans notre application nous pouvons instanciés “**Mongoose**”.
  - C’est lui qui va gérer la connexion à la base MongoDB.

Node, Express, MongoDB

- Voyons un extrait de code « testMongoose.js » initialisant une connexion à MongoDB via Mongoose :

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://@127.0.0.1:27017/gretajs', {useNewUrlParser: true},
  function (err) {
    if (err) {
        throw err;
    } else console.log('Connected');
  }
);
// Schéma définissant une collection
var countriesSchema = new mongoose.Schema({
    _id: {type : mongoose.Schema.ObjectId},
    code: {type: String},
    name: {type: String}
});
// Association entre le schéma et la collection retourne un Model Mongoose
var collection = mongoose.model('Countries', countriesSchema, 'countries');
collection.find(function (err, comms) {
    if (err) {
        throw err;
    }
    console.log(comms);
    mongoose.connection.close();
});
```

- **Au vu du code de connexion avec Mongoose, il semble possible de créer des schémas au format Json :**
  - En effet il est possible de décrire un schéma de mongoose au format JSON dans un fichier texte (.json) plutôt que dans du code JavaScript.
  - Reste à concevoir un module générique qui va initialiser les Schémas au démarrage de l'app en lisant le fichier database_schema.json.

NB : Ce module existe dans OTF² :

`/otf_core/lib/otf_schema_loader.js`

Node, Express, MongoDB

- **Implémentation d'un module de chargement de schéma Mongoose.**
  - Comme nous avons chargé les actions → contrôleurs pour le **dynamicRouter**, nous allons charger le schéma mongoose des collections.
  - Créez un fichier "**database_schema.json**" à la racine du projet sur ce modèle :

```
{
    "Countries" : {
        "collection" : "countries",
        "schema": {
            "_id": "ObjectId",
            "code": "String",
            "name": "String"
        }
    },(…)
}
```

Node, Express, MongoDB

- **Exemple de fichier “database_schemas.json” :**

```
{
    "Exercices" : {
        "collection": "exercices",
        "schema": {
            "_id": "ObjectId",
            "titre": "String",
            "liste": []
        }
    },
    "Countries" : {
       "collection": "countries",
       "schema": {
            "_id": "ObjectId",
           "code": "String",
           "name": "String"
        }
    },

(…)  // → suite colonne de droite
```

```
    "Users": {
        "collection": "users",
        "schema": {
            "_id": "ObjectId",
            "name": "String",
            "firstName": "String",
            "login": "String",
            "mdp": "String",
            "function": "String",
            "office": "String",
            "date_naiss": "String",
            "adresse1": "String",
            "adresse2": "String",
            "cp": "String",
            "city": "String",
            "country": {
                "type": "ObjectId",
                "ref": "Countries"
            },
            "mobile_phone": "String",
            "home_phone": "String"
        }
    }
}
```

Node, Express, MongoDB

- **Nous allons tester le chargement et l'utilisation de nos schémas dans le fichier "testmongoose2.js" :**

```
var fs = require('fs');
global.schemas = {};
var mongoose = require('mongoose');
mongoose.connect('mongodb://simplon:azerty127.0.0.1:27017/simplonjs...', function (err) {
    if (err) {
        throw err;
    } else console.log('Connected');
});
// chargement des schémas depuis le fichier de configuration JSON dans une variable
var database_schemas = JSON.parse(fs.readFileSync("database_schemas.json",'utf8'));
// Initialisation de chaque schéma par association entre le schéma et la collection
for (modelName in database_schemas) {
    global.schemas[modelName] = mongoose.model(modelName,database_schemas[modelName].schema,
              database_schemas[modelName].collection);
}

/* On obtient un tableau de Models à partir des schémas accessible via
 * la variable GLOBAL.schemas qui permettent d'exécuter des requêtes.*/
global.schemas["Countries"].find({code : "FR"}, function (err, comms) {
    if (err) { throw err; }
    // comms est un tableau de hash
    console.log(comms);
    mongoose.connection.close();
});
```

- **Exercice d'intégration de Mongoose :**
  - Modifiez le code de connexion à la base de données dans votre application Express App et utilisez Mongoose
  - Puis intégrez le chargeur de schéma que nous avons testé dans "**testmongoose2.js**" dans l'application, dans "**appdyn.js**"
  - Enfin modifiez les contrôleurs pour qu'ils réalisent les requêtes via les Models Mongoose find(), create() et update() et remove().

- **Exercice correction intégration de Mongoose :**
  - Dans “**appdyn.js**” nous allons modifier la connexion à la base de données :

```
GLOBAL.schemas = {};

// Configuration de la connexion à la base de données via Mongoose :
var mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/simplonjs', function (err) {
    if (err) {throw err;} else console.log('Connected');
});
// chargement des schémas depuis le fichier de configuration database_schemas.json
var database_schemas = JSON.parse(fs.readFileSync("database_schemas.json", 'utf8'));

for (modelName in database_schemas) {
    GLOBAL.schemas[modelName] = mongoose.model(modelName,
         database_schemas[modelName].schema,
         database_schemas[modelName].collection);
}
(…)
```

NB : pensez à commenter en bas du fichier “appdyn.js” la connexion via le driver natif mondodb.

- **Exercice correction intégration de Mongoose :**
  - Voyons la modification des contrôleurs pour utiliser **global.schemas[“<NOM_SCHEMA>”]** exemple avec “**countries.js**” :

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

/* GET list of countries */
router.get('/', function(req, res, next) {
 global.schemas["Countries"].find({}, function(err, result) {
    if (err) {
      throw err;
    }
    //console.log(result);
    res.render('countries', {title: 'Liste countries', country: result});
  });
});
module.exports = router;
```

NB : on utilise la méthode find avec un objet vide pour récupérer l’ensemble des données de la collection countries.

- **Exercice correction intégration de Mongoose : “createUser.js” :**

```
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');
var ObjectId = mongoose.Types.ObjectId;
/* Insert one new user into database. */
router.route('/').get(function (req, res) {
    console.log('req.originalUrl : ', req.originalUrl);
    if (!req.query.hasOwnProperty("_id")) req.query._id = new ObjectId();
    GLOBAL.schemas["Users"].create([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[0]._doc
            });
        } // fin callback de l'insert
    ); // fin de l'insert()
}); // fin de la gestion de la route

module.exports = router;
```

NB : On remarque l’utilisation de mongoose.Type.ObjectId pour créer un _id pour l’enregistrement à inséré. On l’ajoute à req.query que l’on passe en paramètre de la méthode “create([req.query],...”

- **Exercice correction intégration de Mongoose : “formUser.js” :**

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

/* GET user from _id into url */
router.route('/:_id').get(function (req, res) {
    GLOBAL.schemas["Users"].find({_id: new ObjectId(req.params._id)}, 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]
        });
    });
});
module.exports = router;
```

NB : On remarquera la création d’un ObjectId à partir de la chaîne de caractères _id récupérée dans la liste déroulante des utilisateurs.

- **Conclusion intégration Mongoose :**
  - Intégration de Mongoose afin de rendre plus souple et plus dynamique l'intégration de collections dans notre application.
  - L'ajout des configurations JSON décrivant les schémas qui permettront de construire la base de données et les actions permettent plus de souplesse dans la maintenance et les développements futurs.
  - Vous pourriez ajouter des paramètres dans le fichier "**config_actions.json**" pour rendre générique les accès à la base de données.

Node, Express, MongoDB

- **Exercice : Reflexion d'architecture logicielle**
  - Avec l'intégration de Mongoose et des schémas décrit dans un fichier de configuration JSON on peut modifier le code et on obtient pour un find() un code ressemblant a ceci, illustration avec le contrôleur exos.js :

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

/* GET Exercices list. */
router.get('/', function(req, res, next) {
  var type = req.method;
  var path = req.originalUrl;

  GLOBAL.schemas['Exercices'].find({}, function(err, result) {
    if (err) { throw err; }
    console.log(result);
    res.render('exos', {title: 'Express', exos: result[0]});
  });
});
module.exports = router;
```

NB : Trouver comment paramétrer le nom du "Model" (au sens Mongoose) pour l'action considérée...

Node, Express, MongoDB

- **Correction : Reflexion d'architecture logicielle Exemple structure JSON :**

```
(…)  },
    "GET/exos": {
        "controler": "exos",
        "modelName": "Exercices",
        "view": "exos"
    },
(…)
```

- **Ci-dessous le code modifié du contrôleur exos.js**

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

/* GET Exercices list. */
router.get('/', function(req, res, next) {
  var type = req.method;
  var path = req.originalUrl;
  //if (path.split('/').length > 0) path = '/'+path.split('/')[1]
  GLOBAL.schemas[GLOBAL.actions_json[type + path].modelName].find({}, function(err, result) {
    if (err) {
      throw err;
    }
    console.log(result);
    res.render(GLOBAL.actions_json[type + path].view, {title: 'Express', exos: result[0]});
  });
});
module.exports = router;
```