Mongoose

Mongoose est une surcouche à MongoDB pour Node.JS permettant de valider ses données par rapport à des schémas préalablement définis.

Afin d'utiliser convenablement mongoose au sein d'Hapi et de pouvoir gérer facilement les connexions, nous allons passer par les modules `k7` et `k7-mongoose`. Si vous regardez bien `k7`, il s'agit en fait d'un client permettant de gérer de manière transparente, à la configuration, différents moteurs de base de donnée et d'en changer quand bon vous semble (à la manière de PDO en PHP).

Pour intégrer convenablement `k7` au sein de notre projet, nous allons créer le fichier `config/manifest/models.js` suivant :

```
'use strict';
const fs        = require('fs');
const extend    = require('lodash/extend');
const path      = require('path');
const modelsDir = path.join(__dirname, '../../app/models/');
const envConfig = require('../environments/all');
const models    = fs.readdirSync(modelsDir);

module.exports.init = server => {
    return new Promise((resolve, reject) => {
        server.register({
            register : require('k7'),
            options     : extend(envConfig.databases.hapi, {
                models   : [path.join(modelsDir, '**/*.js')],
                adapter  : require(envConfig.databases.hapi.adapter),
                promise  : bluebird,
            }),
        }, err => {
            if (err) {
                reject(err);
                return;
            }
            resolve();
        });
    });
};
```

Ensuite, dans votre fichier d'environnement `all.js`, vous devrez rajouter le bloc suivant :

```
{
  databases : {
   hapi : {
    adapter           : 'k7-mongoose',
    connectionString  :
        `mongodb://${process.env.MONGO_HOST}/${process.env.MONGO_NAME}`,
    connectionOptions : {
      user   : process.env.MONGO_USER,
      pass   : process.env.MONGO_PASSWORD,
      server : {
          auto_reconnect : true,
          socketOptions  : { keepAlive : 1 },
      },
      replset : {
          auto_reconnect : true,
          socketOptions  : { keepAlive : 1 },
      }
    },
  },
 },
}
```

> Ce bloc de configuration permet de setter directement les informations de connexion depuis les variables d'environnement. Ce qui permet de lancer le projet sur différents postes ayant, par exemple, des infos de connexion différentes, sans aucune difficulté

Pour chaque model, vous devrez vous baser sur le modèle suivant dans le répertoire `app/models/` :

```
'use strict';

const jsonToMongoose    = require('json-mongoose');
const mongoose          = require('k7-mongoose').mongoose();

module.exports = jsonToMongoose({
    // réglages du model
});
```

Comme vous pouvez le voir, nous utilisons la library `json-mongoose` permettant de simplifier la création et l'organisation d'un model. Vous trouverez sa documentation ici : https://github.com/Goomeo/json-mongoose. Cette library vous permettra de définir votre schéma de donnée par rapport à un schéma `joi`.

> **Attention** : Dans la documentation d'exemple, il est montré comment utiliser le `_id` en mode autoincrement comme le ferait une base MySQL classique. **N'utilisez surtout pas ce mode**. En plus de ne pas respecter la norme mongo, il ne peut s'utiliser qu'au travers des instances de votre model et non via `monModel.insert({})`.

Migration MongoDB

Il se peut que, lors de l'avancée de votre projet, vous ayez des manipulations à faire dans la BDD associée (ajout de nouvelles entrées, modification des indexes, modification de certaines entrées, ...).

Si vous êtes seul sur votre projet, et qu'il tourne à un seul endroit, cela peut être vite fait. Mais, vous allez très certainement être avec différents environnements et serveurs où vous devrez relancer manuellement vos scripts de migration. Et, qui dit manipulation humaine dit aussi erreurs humaines possibles.

Pour cela on utilise le plugin `hapi-mongo-migration` que vous trouverez ici : https://github.com/Goomeo/hapi-mongo-migration

En ce qui concerne la partie configuration de la connexion MongoDB, on va réutilisé ce que l'on a fait pour K7. Pour le reste, vous devrez rajouter l'entrée du plugin au sein de Hapi avec la configuration suivante (je ne vais pas vous donner toutes les briques) :

```
const appPath = '../../app';

// ...

{
    options  : {
        url             : envConfig.databases.hapi.connectionString,
        user            : envConfig.databases.hapi.connectionOptions.user,
        password        : envConfig.databases.hapi.connectionOptions.pass,
        collection      : 'hapiMigration',
        migrationPath   : path.join(__dirname, appPath, '/migrations'),
    },
}
```

La partie qui nous intéresse est le paramètre `migrationPath` qui indique au plugin où se situent les différents scripts de migration. Ici, ils sont dans `app/migrations`. Vous trouverez des exemples sur le repository du plugin.

> **Note** : Ici, ce n'est pas mongoose qui est utilisé pour l'enregistrement BDD mais directement le connecteur MongoDB. Faites donc attention aux types des données que vous sauvegardez.
>
> Je vous conseille aussi d'appeler vos scripts avec un numéro incrémental en début afin que les scripts soient bien exécutés dans le bon ordre (ex : `01_default_user.js`)

Travail à faire

- Vous devrez donc créer le CRUD complet des utilisateurs au sein d'Hapi avec sauvegarde dans une base de donnée MongoDB en vous basant sur la structure du TP dernier. Bien entendu, vous devrez respecter la norme REST (codes de réponse HTTP et méthodes d'entrées).
- Vous noterez toutefois que vous devrez gérer les cas des champs uniques (un seul email possible à la fois, un seul nir possible par utilisateur, ...).
- Rajoutez de base, via un script de migration, un utilisateur pour que, dès le premier lancement du projet sur un autre poste, on puisse le manipuler via le CRUD.

Le mot de passe devra être encrypté en utilisant votre module créé en TP 2.

Norme REST

Les API REST utilisent les méthodes d'entrées et les codes HTTP pour donner le status des réponses ainsi que, la plupart du temps, le JSON pour l'envoi des données.

Par exemple, pour une API REST sur les utilisateurs vous aurez les requêtes entrantes suivantes :

- `GET /users` : Récupération de tous les utilisateurs ou en fonction de paramètres GET pour filtrer
- `GET /users/:id` : Récupération d'un utilisateur. Si non trouvé retourne une 404
- `POST /users` : Sauvegarde **un nouvel** utilisateur
- `PUT /users/:id` : Modifie **un utilisateur existant**. Si non trouvé retourne une 404
- `DELETE /users/:id` : Supprime **un utilisateur existant**. Si non trouvé retourne une 404

Pour les retours, aidez vous de Hapi-boom et de hapi-boom-decorator en cas d'erreur, il vous les formatera comme il faut.

En ce qui concerne les codes de retour HTTP, en cas de succès vous aurez les codes suivants :

- **200** : Requête effectuée avec succès. Utilisé que pour des requêtes de récupération d'informations.
- **201** : Contenu créé avec succès. Utilisé lors de la sauvegarde ou la modification d'une donnée sur le serveur.
- **204** : Aucun contenu de retourné. Ce statut est utilisé pour les requêtes `DELETE` si elles se sont passées avec succès.

Récapitulatif des avancées du projet

1. Intégrer Mongoose
2. Créer le CRUD des utilisateurs avec liaison BDD
3. Créer une route `/users/generate` permettant de générer 100 nouveaux utilisateurs avec la library Faker.

Tips

- Les fonctions de Handler entrantes (celles appelées par les routes) prennent toujours deux paramètres : `request` et `reply`.
- Vous pouvez récupérer la variable `server` au travers de `request.server` pour, par exemple, accéder à un plugin spécifique.
- Pour accéder à un model mongoose, vous pouvez de la manière suivante : `server.database.monmodel`.
- Les models mongoose retournés via la méthode plus haut vous permettent d'utiliser ses fonctions statiques et magiques (find, findAll, findOne, update, insert, ...).
- Si vous voulez créer une instance de votre model, vous devrez faire `let model = new server.database.monmodel()`.
- Au travers de `json-mongoose` toute fonction du model est promisifiée (vous pouvez faire `model.save().then()`, `mommodel.find().then().catch()`, ...)
- Si vous obtenez l'erreur `model monmodel.js is incorrect`, commentez tous les `require` non utilisés, vérifiez que vous utilisez bien `module.exports` (avec un s) et vérifiez que votre schéma soit correct.