Présentation du projet et Mongoose

Avant de commencer le TP, voici ce que vous allez faire durant les 4 dernières semaines ensembles :

- TP 5 (celui-ci) :
  - Réaliser une API gérant des utilisateurs (ajout, modification, suppression, listage et récupération) au travers d'HAPI
  - Gestion de la persistance des données avec Mongoose
- TP 6 :
  - Envoi, par email à l'utilisateur, de ses identifiants de connexion à la création
  - Ajout d'un moteur de récupération de mot de passe
- TP 7 (sera plus un cours qu'un TP):
  - Présentation du debugger dans webstorm
  - Présentation des différents modes de mise en production
- TP 8 :
  - Présentation de Socket.io
  - Séparation en deux projets distincts la partie gestion des utilisateurs et envoie d'emails (approche micro-services)

Vous devrez donc avoir, à la fin de ces TP, deux projets présents sur Github que vous devrez me donner (avec le lien de chacun d'eux) afin que je puisse vous noter dessus. La note prend en compte le fonctionnement des projets ainsi que l'organisation/apparence du code (bon respect du const/var/let, initialisation des plugins au bon endroit, ...)

Les TPs que vous allez avoir à partir de celui-ci seront que d'un seul par semaine englobant les deux séances que l'on a ensemble.

Mongoose

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

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 path      = require('path');
const modelsDir = path.join(__dirname, '../../app/models/');
const models    = fs.readdirSync(modelsDir);

module.exports.init = server => {
    return new Promise((resolve, reject) => {
        server.register({
            register : require('k7'),
            options  : {
                connectionString : 'mongodb://localhost:27017/hapi',
                adapter          : require('k7-mongoose'),
                models           : [
                    path.join(routeDir, '**/*.js')
                ],
            }
        }, err => {
            if (err) {
                reject(err);
                return;
            }

            resolve();
        });
    });
};
```

Attention : Pensez à déplacer les infos de l'adaptateur utilisé ainsi que l'url de connexion dans les fichiers d'environnement afin de gérer les différentes configurations.

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`.

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, ...).

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

Bonus : Rajouter une url pour insérer automatiquement 100 utilisateurs différents en vous aidant de la library `Faker`.