Atelier 7: Mongoose, JsonWebToken

Consulter la documentation mongoose http://mongoosejs.com/docs/guide.html
Démarrer le serveur BD : **mongod –dbpath *yourdirectorydata***

L'objectif de l'atelier est de developer une application de gestion de taches (todos app), utilisant les tokens pour l'authentification.

Structure du projet:

- *models* : contient les schema et modeles mongoose définis, ainsi que le fichier db.js spécifiant la chaine de connexion a la base de données
- *api* : contient les routes définies pour l'api REST

1. Créer un projet express intitulé todoProject. Créer les repertoires api et models
2. Connexion BD
   - installer le module mongoose: **npm install mongoose –save**
   - créer le fichier *models/db.js* et spécifier la chaine de connexion

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/twin');
```

twin : nom de la base de données

Dans le fichier *app.js*, inclure le module *db.js* et spécifier les routes */api/users* et */api/todos*

```
var db = require('./models/db');
var users = require('./api/users');
var todos = require('./api/todos');
```

3. Schema et model todo

Une tache (todo) est caractérisée par un texte (text:String), une heure et une confirmation de fin de tache (completed : Boolean et completedAt:Number)

- créer les schema et model correspondants

Everything in Mongoose starts with a Schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection. To use our schema definition, we need to convert our blogSchema into a Model we can work with

4. Validation

Consulter la documentation http://mongoosejs.com/docs/validation.html

todo doit verifier les conditions suivantes :

- Le champ *text* obligatoire, non vide et sans espace
- Le champ *completed* est égal a false par défaut
- Le champ *completedAt* doit etre egal a null par défaut

```
var mongoose = require('mongoose');

var todoSchema = new mongoose.Schema({
    text: { type: String, required: true, minlength: 1,trim: true},
    completed: {type: Boolean, default: false},
    completedAt: {type: Number, default: null}
})
module.exports = mongoose.model('Todo', todoSchema);
```

5. Développer l'API Rest (*api/todos.js*):

- Créer une ressource: *POST /todos*

```
router.post('/',function(req,res){

    var todo = new Todo(req.body);
    todo.save(function(err,todo){
        if (err)
            res.send(err);
        else
            res.send(todo);

    })
});
```

- Lister une resource : *GET /todos*

```
router.get('/', function(req,res){

    Todo.find(function(err,todos){
        if(err)
            res.send(err);
        if (!todos)
            res.status(404).send();
        else
            res.json(todos);
    });
});
```

- Récupérer une ressource par id : *GET /todos/:id*

```
//find - findOne - findById
router.get('/:id',function(req,res){
    var id = req.params.id;
   // if (!ObjectID.isValid(id))
     //   res.status(404).send();
    Todo.findById(id).exec(function(err,todo){
        if(err)
            res.status(400).send(err);
        if (!todo)
            res.status(404).send();
        else
            res.json(todo);
    });

});
```

- Supprimer une ressource : *DELETE /todos/:id*

```
//remove -findByIdAndRemove
router.delete('/:id',function(req,res){
    var id = req.params.id;
   // if (!ObjectID.isValid(id))
   //     res.status(404).send();
    Todo.findByIdAndRemove(id, function(err,todo){
        if(err)
            res.send(err);
        else
            res.send();
    })

})
```

- Mettre a jour une ressource : *PUT /todos/:id* : mettre a jour le champ text si spécifié. Si completed est spécifié a true, mettre a jour aussi la valeur de completedAt en spécifiant l'heure de finalisation de la tache

Vous pouvez utiliser ObjectID afin de valider les id specifies dans les URI

```
var ObjectID = require('mongodb').ObjectID;
```

Sous documents:

6. Chaque todo contient une liste de Media

Définir le schema Media caractérisé par les informations suivantes:

name : chaine

uploadDate de type date et qui par défaut est égal a la date systeme

7. Définir l'API REST suivante

GET	/api/todos/1234/medias
POST	/api/todos/1234/medias
PUT	/api/todos/1234/medias
GET	/api/todos/1234/medias/1234
PUT	/api/todos/1234/medias/1234
DELETE	/api/todos/1234/medias/1234

8. Schema/model de l'utilisateur

Un utilisateur est caractérisé par un email et un mot de passe (email:String, password:String)

L'email est unique, obligatoire, non vide et sans espace.

Le mot de passe est aussi obligatoire, contenant au minimum 6 caracteres

```
var userSchema = new mongoose.Schema({
    email: {
        type: String, required: true, trim: true, minlength: 1, unique: true},

        password: {
            type: String, required: true, minlenght: 6
        }
});
module.exports=mongoose.model('User',userSchema);
```

9. Validation personnalisée pour les emails

Afin de valider les emails introduits par l'utilisateur, nous pouvons utiliser les fonctions de validation du module *validator*

- installer le module validator: **npm install validator –save**
- Consulter la documentation http://mongoosejs.com/docs/validation.html et https://www.npmjs.com/package/validator
- Nous utiliserons la function *isEmail* comme suit :

```
var valid = require('validator');
var userSchema = new mongoose.Schema({
    email: {
        type: String, required: true, trim: true, minlength: 1, unique: true,
        validate: {
            validator: valid.isEmail,
            message: '{VALUE} is not a valid email'
        }},
```

10. Register

Les utilisateurs peuvent s'enregister en spécifiant l'email et le mot de passe */api/users/register*

```
router.post('/register',function(req,res){

    var user = new User(req.body)
    user.save(function(err,user){
        if (err)
            res.send(err);
        else
            res.send(user);

    })
});
```

11. Ajouter dans le schema todo un attribute createdBy qui reference l'utilisateur ayant crée le todo,

Tester POST /api/todos, utiliser populate afin de récupérer les informations de l'utilisateur dans le GET /api/todos

12. Hachage du mot

Pour ne pas stocker le mot de passe en clair dans la base de données, nous allons utiliser le module *bcrypt* pour le hachage du mot de passe (les algorithmes les plus connus pour le hachage du mot de passe sont pbkdf2 et bcrypt)

Pour plus d'informations, consulter https://gooroo.io/GoorooTHINK/Article/13023/The-difference-between-encryption-hashing-and-salting/2085#.WOKQPlM1_EY et http://dustwell.com/how-to-handle-passwords-bcrypt.html

- installer le module bcrypt-nodejs : **npm install** bcrypt-nodejs **–save**
- Consulter la documentation https://www.npmjs.com/package/bcrypt-nodejs
- utiliser les fonctions hashSync et genSaltSync comme suit :

```
router.post('/register',function(req,res){
    var user = new User({
        password: bcrypt.hashSync(req.body.password, bcrypt.genSaltSync(10)),
        email : req.body.email
    })
    user.save(function(err,user){
        if (err)
            res.send(err);
        else
            res.send(user);

    })
});
```

13. Login

Pour s'authentifier, l'utilisateur spécifie son email et son mot de passe. La recherche se fait par email (unique). Ensuite, en cas de suuces, le password fourni est comparé avec la valeur hachée stockée dans la BD, et ce en utilisant la function *compareSync* du module *bcrypt*

```
router.post('/login',function(req,res){

    User.findOne({email:req.body.email},function(err,user) {
            if (err)
                res.send(err);
            if (!user)
                res.status(401).json(user);
            else {
                if (bcrypt.compareSync(req.body.password, user.password)) {
                    res.status(200).json(user);

                }
```

14. JWT

Nous utiliserons JsonWebToken pour authoriser les utilisateurs a effectuer les operations si leurs parametres d'authentification sont validés (alternative aux sessions plus adequate avec les Rest API).

- installer le module jsonwebtoken : **npm install jsonwebtoken –save**
- consulter la documentation https://jwt.io

- modifier la function ci-dessous. Si l'email et le mot de passe sont validés, un token est généré avec sign(claims,secretkey,expirationtime) comme suit. Le token est renvoyé ensuite au client afin de l'utiliser pour les prochaines requetes

```
var jwt = require('jsonwebtoken');
router.post('/login',function(req,res) {
    ..........

    if(bcrypt.compareSync(req.body.password, user.password))
    {
        console.log('User found', user);
        var token = jwt.sign({email: user.email}, 's3cr3t', {expiresIn: 3600});
        res.status(200).json({success: true, token: token});
    }
    else
    {
        res.status(401).json('Unauthorized');
    }
});
```

- nous allons créer un middleware *authenticate* qui va etre utilisé pour toutes les requetes protégées. Dans le fichier *api/auth.js,* ajouter la function authenticate permettant de verifier si le token est envoyé dans le header de la requete, de verifier ce token avec la function verify de jwt, et en cas de success d'exécuter la prochaine instruction avec next()

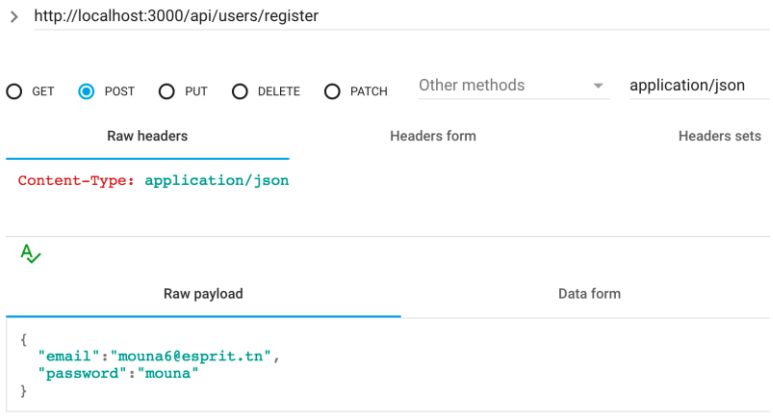
```
var jwt = require('jsonwebtoken');
module.exports.authenticate=function(req, res,next) {
        var headerExists = req.headers.authorization;
        if (headerExists) {
            var token = req.headers.authorization.split(' ')[1]; //
            jwt.verify(token, 's3cr3t', function(err, decoded) {
                if (err) {
                    console.log(err);
                    res.status(401).json('Unauthorized');
                } else {
                    req.user = decoded.email;
                    console.log(decoded.email);
                    next();
                }
            });
        } else {
            res.status(403).json('No token provided');
        }
    };
```

- pour tester cela, nous allons faire en sorte que l'utilisateur doit etre authentifié avant de pouvoir lister les todos existants. Pour cela, dans le fichier todos.js, et dans la route GET /api/todos, ajouter l'appel a authenticate comme suit :

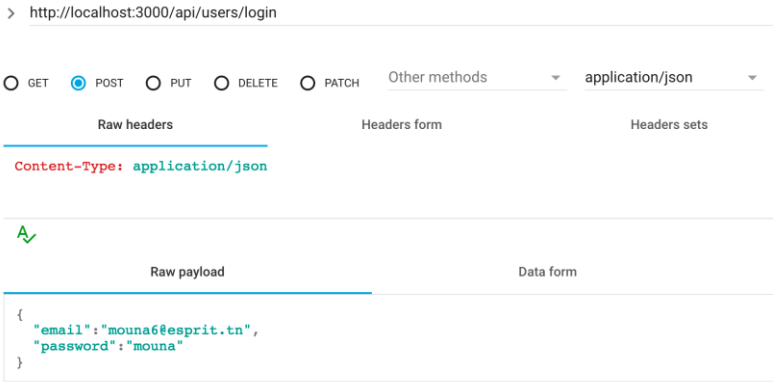
```
var auth = require('../api/auth');

router.get('/', authenticate, function(req,res){

    Todo.find(function(err,todos){
        if(err)
```

- test final avec un client rest

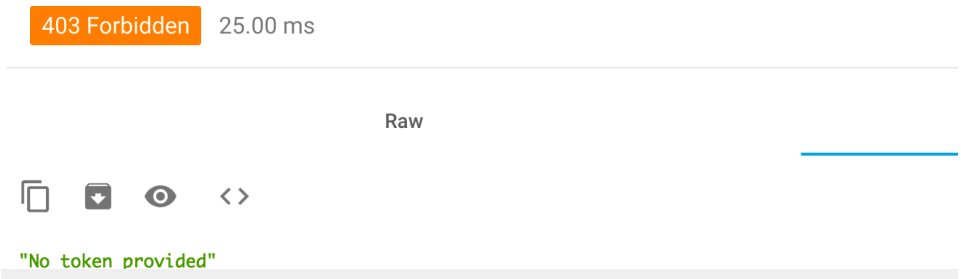
register :



login :



tester GET /api/todos :



ajouter le token dans le header (Authorization) et re tester GET /api/todos :