# Utiliser MongoDB avec Node.js grâce à Mongoose

**Mongoose** est un module Node.js qui s'installe avec **NPM** (Node Package Manager).  
Placez-vous à la racine de votre projet et faites :

***$***> npm install mongoose

Cette commande va installer le module mongoose dans le dossier node\_modules/ (s'il n'existe pas il sera créé automatiquement).

Bien entendu, vu que Mongoose utilise MongoDB, il faut au préalable avoir installé MongoDB et avoir lancé le serveur sur un terminal.

Afin d'utiliser le module de Mongoose, il suffit de l'inclure comme ceci :

[**var**](https://developer.mozilla.org/fr/search?q=var) mongoose = require('mongoose');

Ensuite nous devons préciser sur quelle base de données nous allons travailler :

mongoose.connect('mongodb://localhost/blog', function(err) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

});

Ici la base de données se nomme « blog », si elle n'existe pas elle sera créée automatiquement si une insertion est faite. La méthode connect() va créer un objet Connection accessible via mongoose.connection.

Si vous voulez vous connecter à plusieurs bases de données, il faudra rajouter mongoose.createConnection(‘mongodb://localhost/blog2?); pour chaque nouvelle base de données utilisée (cette méthode renvoie aussi un objet Connection).

Pour se déconnecter de MongoDB après avoir fait tous vos traitements, il faut appeler la méthode close() pour chaque connexion ouverte :

// Si on a utilisé mongoose.connect()

mongoose.connection.close();

Vous ne devez vous déconnecter de la base de données qu'après avoir effectué tous vos traitements dans celle-ci. Comme Node.js fournit un environnement asynchrone, l'ajout en base de données est asynchrone lui aussi. Un exemple est fourni dans ex1.js pour vous montrer comment vous déconnecter correctement de MongoDB après avoir effectué vos traitements.

Mongoose utilise des « Schema » pour modéliser les données. Il permet de définir les types de variables et de structurer vos données (un peu comme si vous définissiez vos tables en SQL).

Pour créer un Schema avec Mongoose, il suffit d'utiliser l'objet Schema :

[**var**](https://developer.mozilla.org/fr/search?q=var) commentaireArticleSchema = [**new**](https://developer.mozilla.org/fr/search?q=new) mongoose.Schema({

pseudo : { type : [String](https://developer.mozilla.org/fr/search?q=string), match: /^[a-zA-Z0-9-\_]+$/ },

contenu : [String](https://developer.mozilla.org/fr/search?q=string),

date : { type : [Date](https://developer.mozilla.org/fr/search?q=date), default : [Date](https://developer.mozilla.org/fr/search?q=date).now }

});

Comme vous pouvez le voir, on peut définir avec plus ou moins de précision le format de donnée que l'on attend pour chaque « champ », cela permet de sécuriser facilement les données que l'on ajoute

Vous pouvez aussi faire en sorte que des valeurs soient aussi des schémas, voici un exemple :

[**var**](https://developer.mozilla.org/fr/search?q=var) articleSchema = [**new**](https://developer.mozilla.org/fr/search?q=new) mongoose.Schema({

auteur : Schema.ObjectId,

contenu : [String](https://developer.mozilla.org/fr/search?q=string),

date : { type : [Date](https://developer.mozilla.org/fr/search?q=date), default : [Date](https://developer.mozilla.org/fr/search?q=date).now },

commentaires : [ commentaireArticleSchema ],

votes : {

plus : { type : [Number](https://developer.mozilla.org/fr/search?q=number), min : 0 },

moins : { type : [Number](https://developer.mozilla.org/fr/search?q=number), min : 0 }

}

});

Ici on associe la clé « commentaires » à un tableau qui aura comme schéma celui des commentaires. Quant à la clé « votes », c'est tout simplement un *hash* qui contient d'autres *hash*… merci à MongoDB de nous permettre une telle flexibilité !

Maintenant que nous avons notre schéma de définition, nous allons maintenant créer un model qui va nous permettre d'insérer des données dans MongoDB en respectant le schéma précisé et d'aller faire des requêtes dessus !

Pour créer un model, rien de plus simple :

[**var**](https://developer.mozilla.org/fr/search?q=var) CommentaireArticleModel = mongoose.model('commentaires', commentaireArticleSchema);

Cela va créer un model nommé « commentaires » à partir du schéma CommentaireArticleSchema et qui va le renvoyer dans la variable CommentaireArticleModel. On peut bien entendu créer le model dans un fichier .js en incluant Mongoose et en le récupérant dans un autre fichier en appelant :

// Récupérer un Model

[**var**](https://developer.mozilla.org/fr/search?q=var) CommentaireArticleModel = mongoose.model('commentaires');

Le model va créer une collection automatiquement nommée « commentaires » (si elle n'existe pas dès qu'un élément sera inséré dedans). Le model est comme une table en SQL si vous aimez les comparaisons.

Pour créer une instance de Model, il suffit de faire :

[**var**](https://developer.mozilla.org/fr/search?q=var) monCommentaire = [**new**](https://developer.mozilla.org/fr/search?q=new) CommentaireArticleModel({ pseudo : 'Atinux' });

// On rajoute le contenu du commentaire, possible de le faire lors de l'instanciation

monCommentaire.contenu = 'Salut, super article sur Mongoose !';

Cela va donc nous créer une instance du Model CommentaireArticleModel, cependant il n'est pas encore sauvegardé dans MongoDB, pour cela il suffit d'appeler la méthode save() de l'instance créée. Cette fonction prend une fonction qu'elle appellera après l'insertion dans la base en donnant comme paramètre l'erreur s'il y en a eu une.

monCommentaire.save(function (err) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

console.log('Commentaire ajouté avec succès !');

});

Voici un petit récapitulatif du code que vous pouvez tester directement :

// Inclusion de Mongoose

[**var**](https://developer.mozilla.org/fr/search?q=var) mongoose = require('mongoose');

// On se connecte à la base de données

// N'oubliez pas de lancer ~/mongodb/bin/mongod dans un terminal !

mongoose.connect('mongodb://localhost/blog', function(err) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

});

// Création du schéma pour les commentaires

[**var**](https://developer.mozilla.org/fr/search?q=var) commentaireArticleSchema = [**new**](https://developer.mozilla.org/fr/search?q=new) mongoose.Schema({

pseudo : { type : [String](https://developer.mozilla.org/fr/search?q=string), match: /^[a-zA-Z0-9-\_]+$/ },

contenu : [String](https://developer.mozilla.org/fr/search?q=string),

date : { type : [Date](https://developer.mozilla.org/fr/search?q=date), default : [Date](https://developer.mozilla.org/fr/search?q=date).now }

});

// Création du Model pour les commentaires

[**var**](https://developer.mozilla.org/fr/search?q=var) CommentaireArticleModel = mongoose.model('commentaires', commentaireArticleSchema);

// On crée une instance du Model

[**var**](https://developer.mozilla.org/fr/search?q=var) monCommentaire = [**new**](https://developer.mozilla.org/fr/search?q=new) CommentaireArticleModel({ pseudo : 'Atinux' });

monCommentaire.contenu = 'Salut, super article sur Mongoose !';

// On le sauvegarde dans MongoDB !

monCommentaire.save(function (err) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

console.log('Commentaire ajouté avec succès !');

// On se déconnecte de MongoDB maintenant

mongoose.connection.close();

});

Exécutez le fichier avec Node.js :

***$***> node ex1.js

Commentaire ajouté avec succès !

***$***>

Maintenant nous allons voir si ces données ont bien été insérées dans MongoDB. Ouvrez un terminal et lancez le binaire mongo (si vous avez suivi mon tutoriel sur MongoDB, il se trouve ici : ~/mongodb/bin/mongo).

> use blog

switched to db blog

> show collections

commentaires

system.indexes

> db.commentaires.find()

{ « contenu » : « Salut, super article sur Mongoose ! », « pseudo » : « Atinux », « \_id » : ObjectId(« 4e995dab2ea479d40f000001?), « date » : ISODate(« 2011-10-15T10:17:15.829Z ») }

>

Comme vous pouvez le voir notre article a bien été inséré dans la collection « commentaires » dans la base « blog » !

Bon c'est bien beau de voir nos données avec le CLI de MongoDB mais maintenant il faudrait pouvoir les récupérer avec Mongoose si on veut faire de vraies applications Web !

Nous allons utiliser directement notre Model « CommentaireArticleModel » qui est un accès direct à la collection « commentaires ». Pour récupérer nos commentaires, nous avons trois méthodes possibles : find, findOne et findById.

C'est la méthode find(query, [fields], [options], callback) que je vais prendre comme exemple ici :

CommentaireArticleModel.find([null](https://developer.mozilla.org/fr/search?q=null), function (err, comms) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

// comms est un tableau de hash

console.log(comms);

});

Ici je donne null comme *query*, donc Mongoose va aller chercher toutes les entrées de la collection « commentaires », ce sont les même requêtes qu'avec la fonction find() de MongoDB. Vous pouvez donc faire par exemple find({ pseudo : ‘Atinux' }) pour n'avoir accès qu'aux commentaires fait par Atinux.

À noter que ces méthodes renvoient un objet Query et que l'on peut continuer la requête si l'on ne précise pas de *callback* à la fonction retournant cet objet. Ensuite il faudra appeler la méthode exec() de cet objet Query afin de l'exécuter et de récupérer les résultats :

// Ici on va récupérer les 3 premiers commentaires ayant le pseudo Atinux

[**var**](https://developer.mozilla.org/fr/search?q=var) query = CommentaireArticleModel.find([null](https://developer.mozilla.org/fr/search?q=null" \t "_blank));

query.where('pseudo', 'Atinux');

query.limit(3);

// peut s'écrire aussi query.where('pseudo', 'Atinux').limit(3);

query.exec(function (err, comms) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

// On va parcourir le résultat et les afficher joliment

[**var**](https://developer.mozilla.org/fr/search?q=var) comm;

for ([**var**](https://developer.mozilla.org/fr/search?q=var) i = 0, l = comms.length; i < l; i++) {

comm = comms[i];

console.log('------------------------------');

console.log('Pseudo : ' + comm.pseudo);

console.log('Commentaire : ' + comm.contenu);

console.log('Date : ' + comm.date);

console.log('ID : ' + comm.\_id);

console.log('------------------------------');

}

});

Il suffit d'appeler la méthode Model.update(conditions, update, [options], callback). Par exemple pour changer tous les pseudos des commentaires étant égal à Atinux par Nikita, nous devons faire :

CommentaireArticleModel.update({ pseudo : 'Atinux'}, { pseudo : 'Nikita' }, { multi : [true](https://developer.mozilla.org/fr/search?q=true) }, function (err) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

console.log('Pseudos modifiés !');

});

Si nous ne précisons pas l'option { multi : true }, Mongoose ne modifiera que la première entrée qu'il trouvera.

Il faut utiliser la méthode Model.remove(conditions, callback).

Pour supprimer tous les commentaires ayant comme pseudo Nikita :

Model.remove({ pseudo : 'Nikita' }, function (err) {

if (err) { [**throw**](https://developer.mozilla.org/fr/search?q=throw) err; }

console.log('Commentaires avec pseudo Nikita supprimés !');

});

Source :

https://atinux.developpez.com/tutoriels/javascript/mongodb-nodejs-mongoose/