Projet Mongo

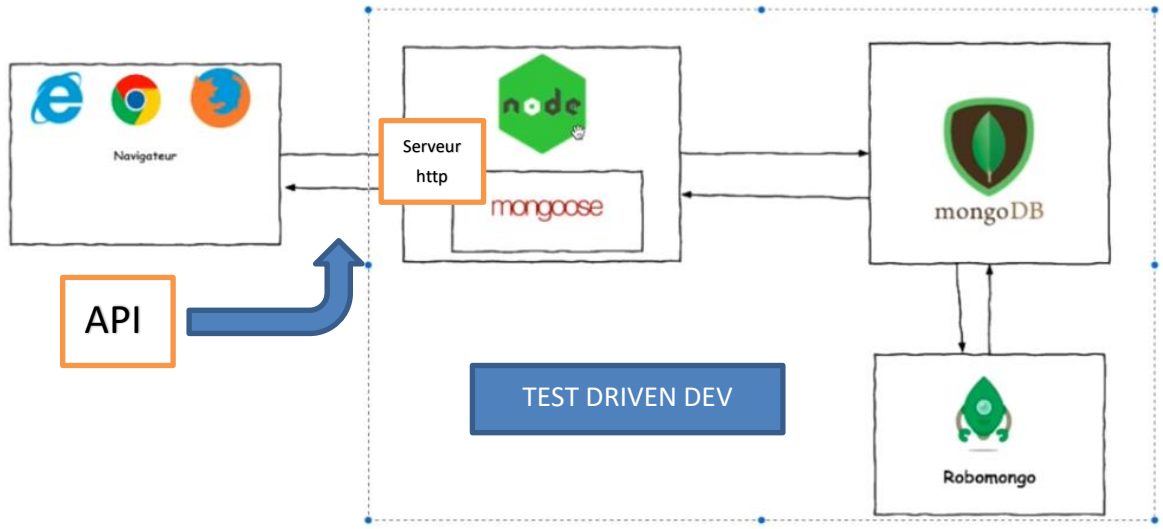
E5 Estiam

Présentation du projet

Le projet MongoDB en local consiste à développer une API afin de gérer une base de données MongoDB sur un site qui gère une vidéothèque. On va donc créer des modèles « utilisateurs » et « films »

Ceci ne sera possible qu'après avoir ajouté un serveur http au serveur NodeJS afin de satisfaire les requête http de votre navigateur.

Mise en place d'un serveur http : Le serveur http sera gérer par le serveur NodeJS et implémentera les réponses aux routes (voir ci-après)



Création de routes

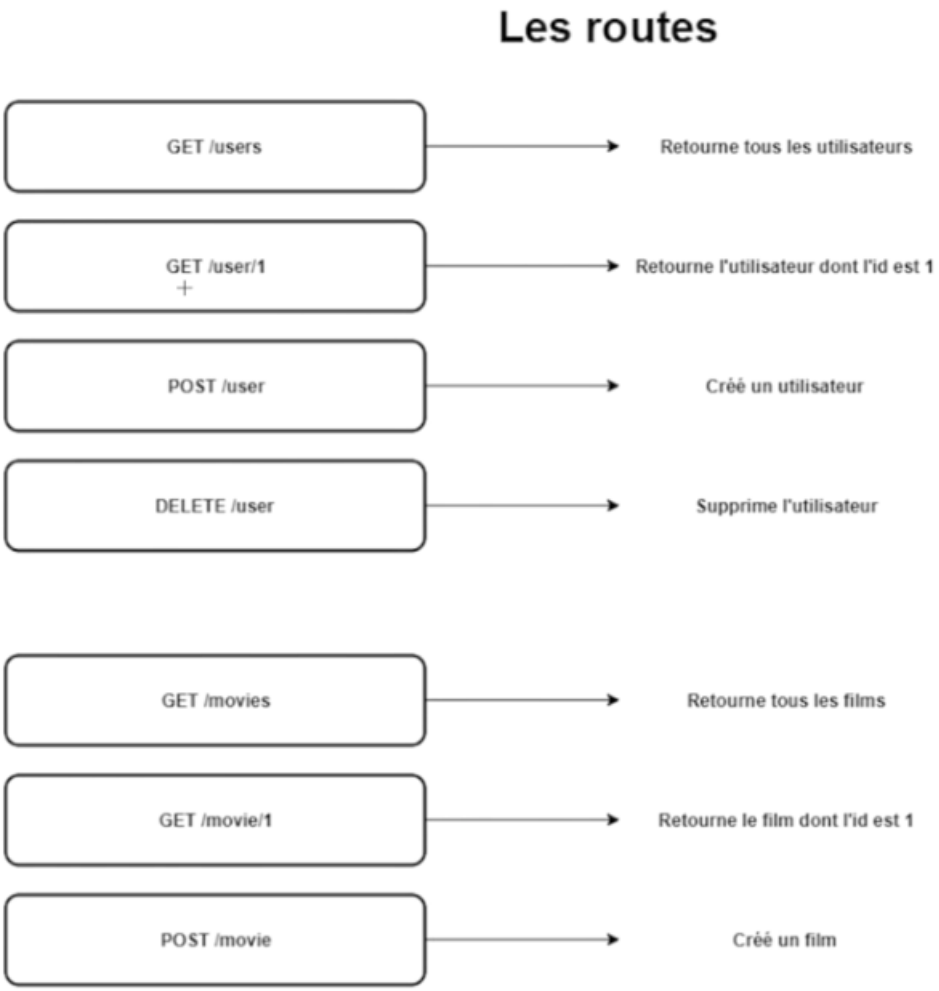
Les routes sont des formats d'URL adressant un API afin de créer ou récupérer des données. Nous allons nous intéresser à 3 types de routes : GET, POST et DELETE.

Les routes de type « GET » peuvent être envoyées pas un navigateur mais pas les routes de type POST et DELETE. Vous devrez utiliser un outil spécifique pour les tester. POSTMAN est un outil très utilisé dans ce cas.

Type de routes	Outil
GET	Votre navigateur
POST / DELETE	POSTMAN

L'objectif de ce projet est de développer l'API en développant les routes afin de populer une base de données MongoDB en local.

Voici les routes qui nous intéressent pour ce projet :



Le projet vous guide pas à pas pour mettre en place votre environnement jusqu'au premières routes qui sont :

- GET /users
- GET /user/1

Mise en place de votre environnement de développement

Mise en place du serveur http avec express

Pour commencer, il faut créer un nouveau dossier et l'initialiser avec « npm ». Creez un dossier mongo-api à l'endroit de votre choix et sur le terminal lancez les commandes suivantes.

⇨ « npm init » Ceci créera le *package.json*
⇨ « npm install –save mongoose@4.11.9 espress » avec ***mongoose*** en version 4.11.9 et ***express*** pour le serveur http. Ceci créera le node_module

Le serveur http ***express*** est là pour écouter et relayer les demandes au serveur NodeJS. Express permet de mettre un serveur http en écoute sur notre serveur Node très simplement

Créez un fichier *server.js* que vous commencerez à remplir avec le code suivant :

```
const express = require('express') ;
const serveur = express() ;
server.listen(3050, () => {
        Console.log(« Ecoute sur le port 3050 »),
    })
```

Ensuite executez la commande ***node server.js*** dans le dossier contenant le fichier. Vous constatez que le serveur écoute sur le port mentionné. Nous sommes prêts à développer ce serveur.

Réponse du serveur http

Ce serveur écoute mais ne répond pas. Enrichissons donc le serveur afin qu'il réponde sur une requête GET. Ainsi, il faut coder dans le fichier *server.js* les actions sur la réception de la requête GET

Nous voulons pour notre exemple que le serveur réagisse sur l'URL suivante :

http://adresse_serveur:port/bonjour

Pour cela ajouter la fonction suivante avant la fonction ***server.listen*** :

```
server.get(« /bonjour », (reg,res) => {
        console.log(« Salut ») ;
})
```

Test du serveur http

Pour tester, cette fois, on ne va pas simplement lancer la commande sur le terminal. On va utiliser son navigateur. En effet, un navigateur peut exécuter une requête GET. Mais le retour se fera, pour l'instant toujours sur le terminal avec notre code en l'état.

Ouvrez donc votre navigateur et testez le code avec l'URL : ***http://localhost:3050/bonjour***.

⇨ Constatez que sur notre terminal vous avez bien le retour console.

Si vous avez bien la chaine de caractère Salut sur votre console c'est que votre serveur http vous répond. On va donc tenter de faire répondre le serveur http non pas sur la console mais sur le navigateur.

On va donc passer le code précédent en commentaire et utiliser la fonction suivante :

```
req.send({result : « reponse du serveur »} ;
```

Attention : A chaque modification de votre fichier server.js vous devez relancer la commande node server.js

Optimation avec le package nodemon

Maintenant que notre server fonctionne, nous allons structurer notre code et faire en sorte que notre server redémarre automatiquement au lieu de le relancer à la main la commande

Installons-le package nodemon avec la commande

⇨ ***« npm install nodemon -g»***

Appelez ensuite *server.js* avec nodemon de la manière suivante

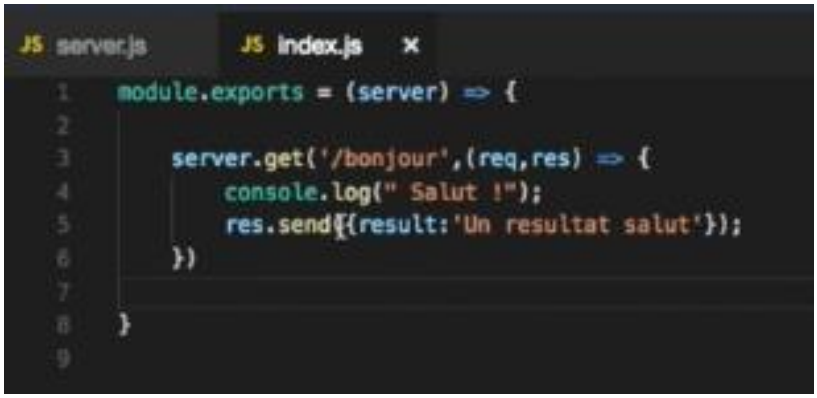
⇨ ***« nodemon server.js »***

Grâce à cette commande si vous modifiez votre fichier *server.js* vous n'aurez plus besoin de relancer la commande. Cela se fera automatiquement. Vous pourrez ainsi consulter le navigateur sans avoir à redémarrer le serveur http.

Organisation de votre code

Pour organiser le code, nous allons séparer les fonctions routes sur un fichier *index.js* dans un dossier route. Créez un dossier « routes » dans votre dossier « mongo-api » et créez un fichier *index.js*. Récupérer le code de la fonction ***get*** que vous aviez mis dans le fichier *server.js* et ajoutez le dans votre nouveau fichier.

Votre fichier *index.js* doit ressembler à cela :



Vous pouvez retester ! ça fonctionne toujours aussi bien !

Mise en place des 2 premières routes « users » et « user »

Finis avec les « bonjour » Nous allons maintenant commencer à écrire les routes dans ce même fichier. Le code devient ainsi le suivant. Nous commençons par ***« users »*** et ***« user »***. Modifiez votre code de la manière suivante

Explications :

La première fonction permet d'avoir une réponse du serveur sur l'URL : http://localhost :3050/users

Par exemple on peut récupérer la liste de tous les utilisateurs sur notre base de données Mongo.

La deuxième fonction permet de récupérer la réponse du serveur sur l'URL :
http://localhost :3050/user/id

« id » precise l'ID d'un user en particulier. Nous pouvons par exemple récupérer toutes les caractéristiques d'un utilisateur dans la base de données Mongo

Les caractéristiques de l'utilisateur sont stockées dans « params ». C'est pourquoi nous avons passé en paramètre « req.params.id »

Testez donc avec les URL suivantes

- http://localhost :3050/user/1
- http://localhost :3050/user/2

Pour bien comprendre ce qu'il se passe, vous allez modifier encore une fois le code :

Retestez donc avec les URL suivantes :

- http://localhost :3050/user/1
- http://localhost :3050/user/2

Vous constatez que le retour du serveur est différent, ce qui vous permet de mieux comprendre le code précédent.

Mise en place des contrôleurs

Vous avez mis en place les routes

Nous allons mettre en place un contrôleur. L'objectif d'un contrôleur est de juguler les demandes. Nous commençons à basculer vers une architecture MVC (Modèle View Contrôleur) dont le but est d'organiser le code. Nous allons donc juste déplacer du code vers un fichier pour distinguer le rôle de chaque partie.

Créez un dossier ***« controlleur »*** dans votre dossier « ***mongo-api »*** et créez le 2 fichiers suivants :

- *user-controller.js*
- *movie-controller.js*

Dans le fichier route, nous appellerons donc les fonctions des contrôleurs qui eux se chargeront de faire la requête.

Voici le code du contrôleur

Comme vous pouvez le remarquer le code dans les fonctions ***« getUsers »*** et ***« getUser »*** provient du fichier *index.js*

En effet comme expliqué ci-dessus, l'intérêt est d'organiser le code. Maintenant dans le fichier *index.js* nous allons appeler les fonctions du contrôleur afin d'exécuter les routes.

Voici le nouveau code du fichier *index.js*

Attention : La ligne de code server.get('/users', UserControlleur.getUsers) ;

peut paraitre complexe mais c'est simplement la simplification de :

Server.get('/users', (req, res) => {

UserController.getUsers(req, res) ;

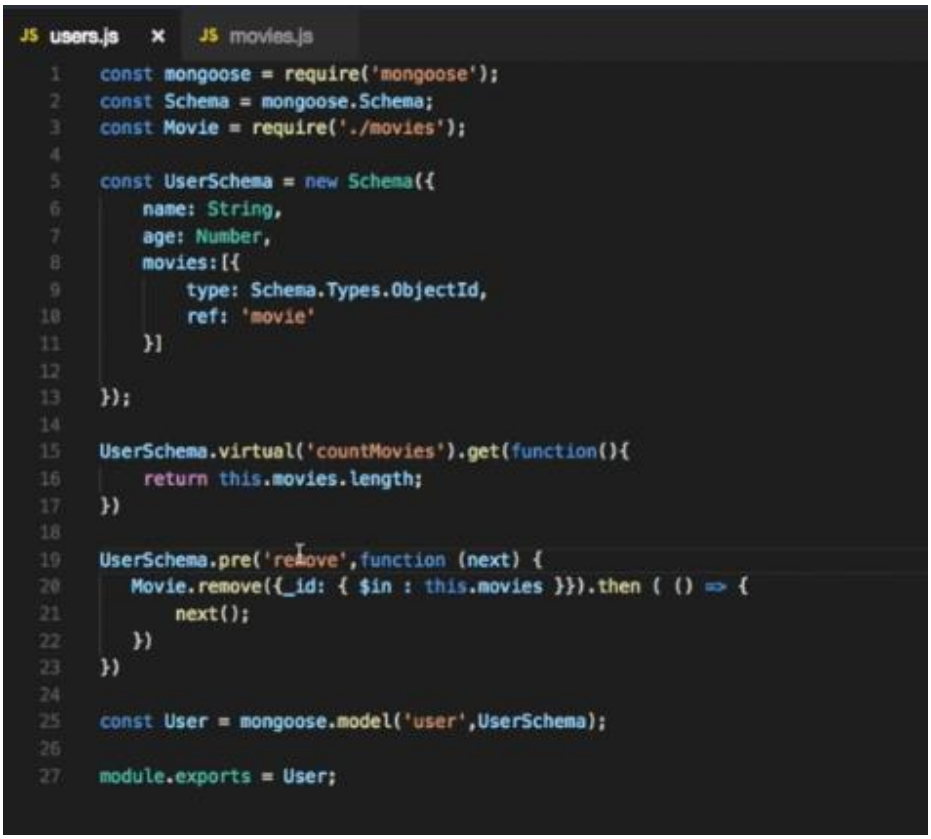
}

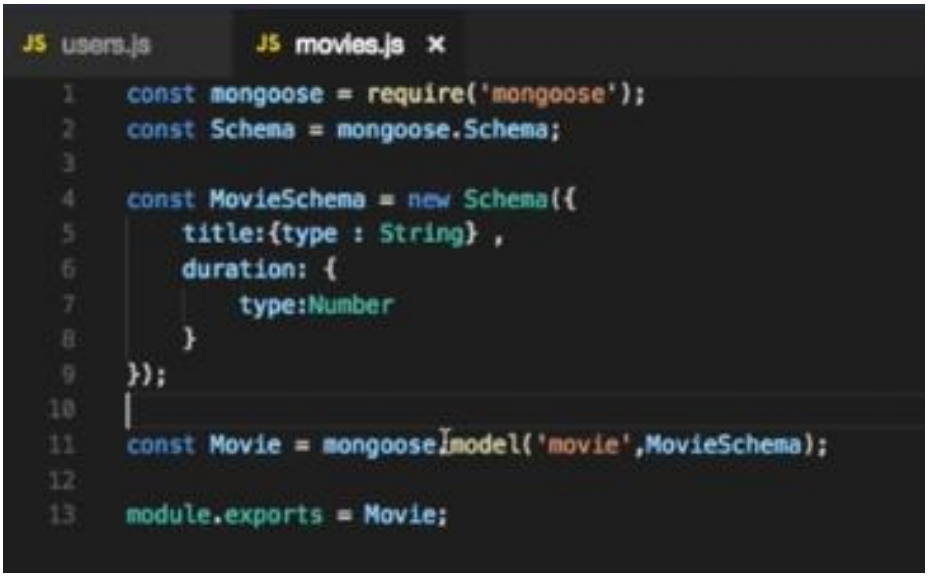
Testez votre code en relançant la commande => « ***nodemon server.js »*** et constatez que tous fonctionnent toujours comme prévu avec votre navigateur

Mise en place des modèles

Continuons notre organisation du code. Créez un dossier ***« models »*** dans votre dossier ***« mongo-api »***

Créez un fichier *users.js* et un fichier *movies.js*. Voici le code de chaque fichier





Postman et requête POST

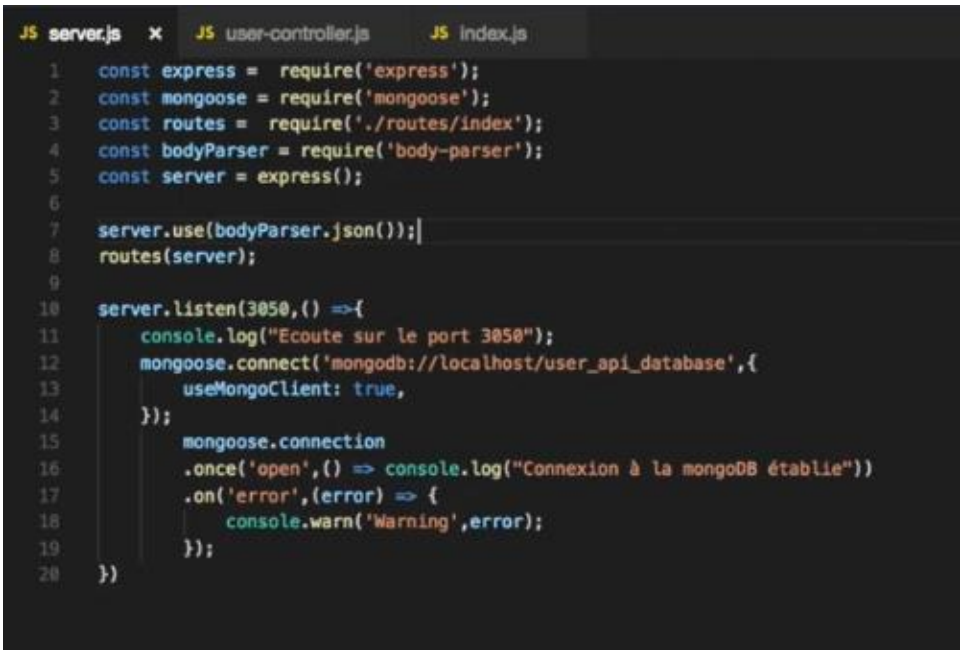
On a mis en place un projet structuré et notre code fonctionne. Avant de commencer son projet, il faut comprendre comment fonctionne la requête POST.

Comme évoqué au début du projet, les requêtes POST et DELETE, qui ont vocation à modifier une base de données ne peuvent être faites à partir d'un navigateur. Ainsi nous allons utiliser un outil dédié çà cela. Cet outil s'appelle POSTMAN

Installer postman en tant qu'extension sur chrome ou en le téléchargeant ici

Avant de commencer nous allons adaptons notre fichier *server.js* remplacer les fonctions ***getUsers*** et ***getUser*** dans le contrôleur des utilisateurs par ***readAll*** et ***read*** afin d'être conforme aux conventions du CRUD (Create, Read, Update, Delete). Nous allons aussi en profiter pour ajouter la route ***« create »*** dans le contrôleur.

Voici les 2 fichiers :



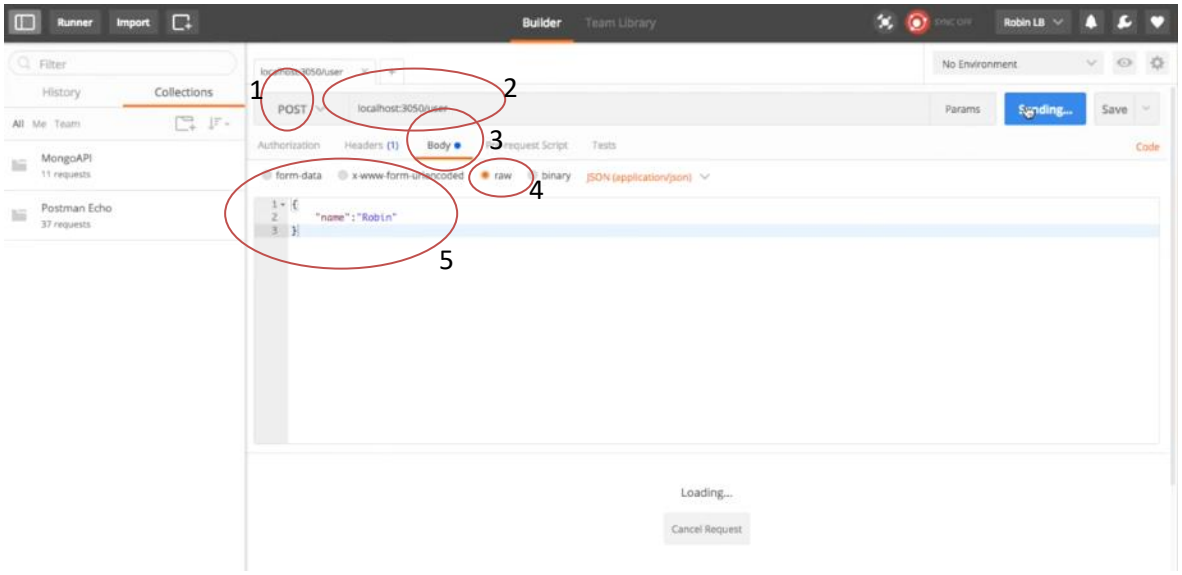
Dans le fichier *user.controller.js* on voit qu'on utilise « ***console.log(body) ; »*** c'est justement les données que l'on va envoyer par POSTMAN comme nous allons le voir ci -après.

On n'oubliera pas d'adapter le fichier *index.js* puisque le nom des routes a été modifié

Et justement dans ce fichier, il manquera la route create qui vient d'être implémentée pour le CREATE du CRUD. Le fichier index.js ressemble donc à cela :

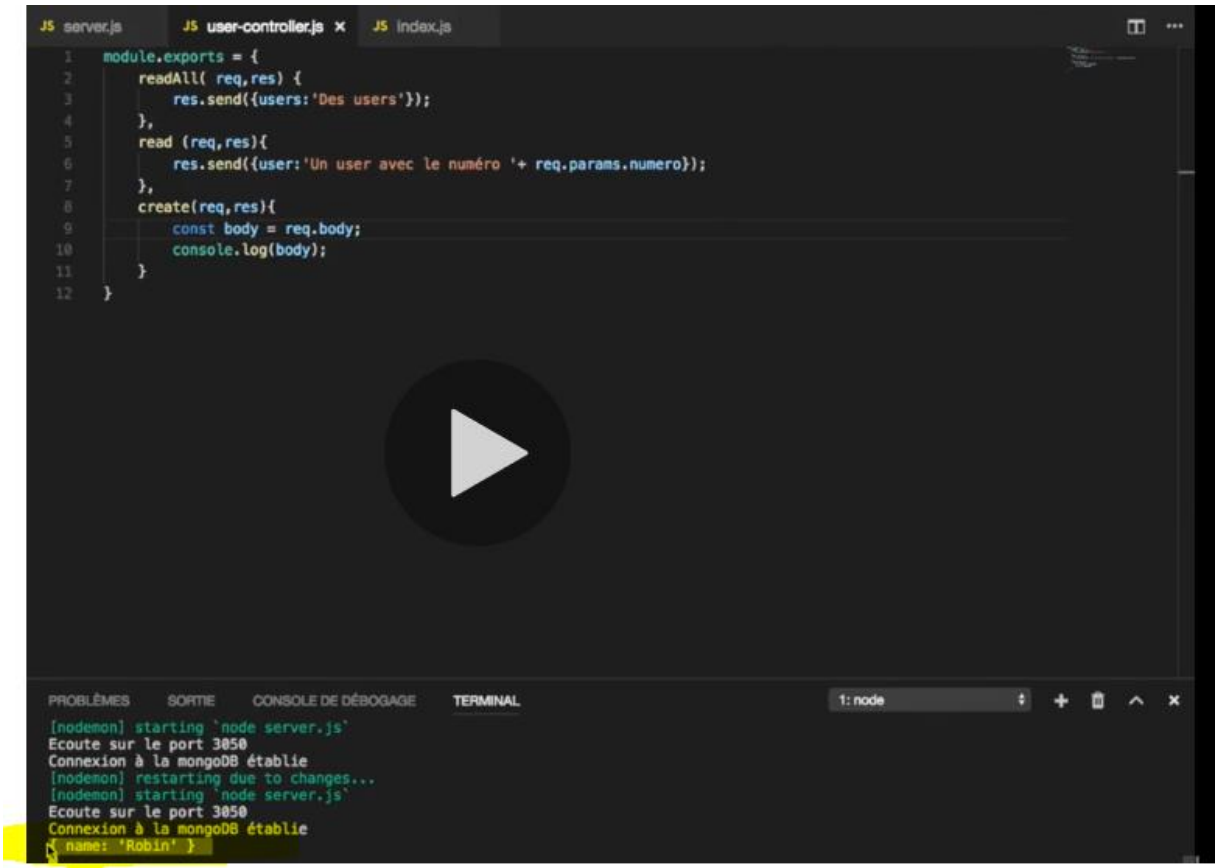
On est prêt à utiliser POSTMAN

Voici comment l'utiliser :

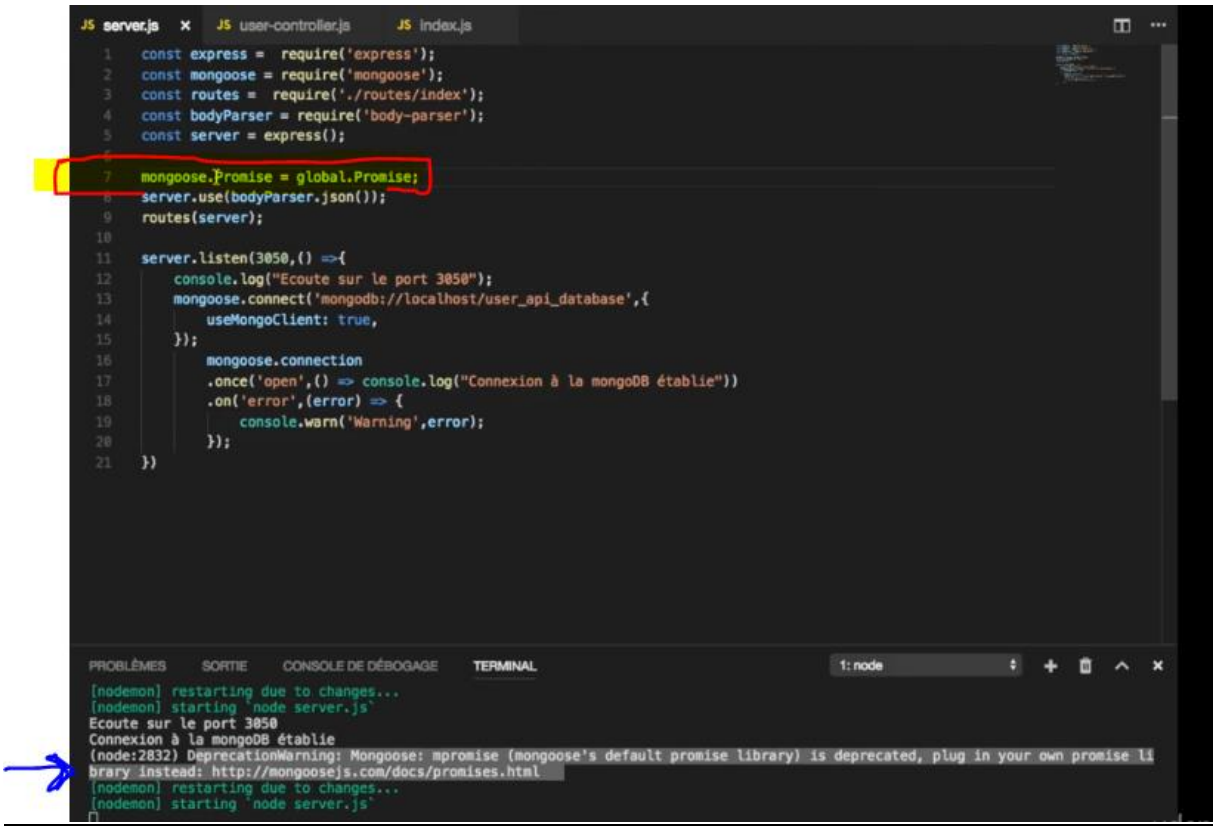


1. On précise le type de requête, ici POST, d'ailleurs à partir de maintenant vous pourrez faire vos requête GET avec POSTMAN puisqu'on peut faire tous type de requête
2. On précise l'URL du serveur avec le port
3. On sélectionne body, c'est le corps du message à envoyer
4. On sélectionne RAW pour envoyer en JSON
5. On écrit les données à envoyer à la base de données

Voici le résultat sur votre console. En effet, comme dut plus faut, nous avons utiliser l'appelle de fonction « ***console.log(body) ; »*** donc le résultat de l'appel de cette requête arrive sur la console. C'est justement tout l'intérêt du projet de faire la jonction avec la base de données. Mais à partir de maintenant c'est à vous de jouer !



En cas de warning comme indiqué en bleu, il faut ajouter le code en rouge dans le fichier *server.js*



Cette erreur est simplement un warning indiquant qu'il faut utiliser le système de promesse de ES6 et non celui de mongoose qui est moins performant.

Barème de notation

Votre projet est prêt à être implémenter. Vous devez développer les routes du CRUD pour les contrôleurs users et pour les movie sans oublier d'adapter les autres fichiers concernés.

Les objectifs sont détaillés ci-après avec un barème correspondant.

1/ Monter son API 5

- Suivre le document et faire fonctionner son projet avec POSTMAN

2/ Finaliser son API et alimenter la base de données MongoDb

- Marier les 2 premières routes jusqu'à la base de données
- Faire le maximum de routes
- Marier les routes restantes

3/ Alimenter sa base de données grâce au code mise en place

Une fois que vous avez implémenter votre projet, il vous suffit de l'enrichir en créant des utilisateurs en quantité. Vous pourrez utiliser une algorithmique adaptée à cette création.

4/ Présentation

5/ Bonus améliorer le front enrichir les modele, schema, document

Fonctionnalités à implémenter	Points
Monter son API	2
Finaliser son API et alimenter la base de données MongoDb	5
Alimenter sa base de données	3
Présentation	2
Bonus : améliorer son front	+4