TD 8 / TME 8 – MongoDB

TD 8 / TME 8 – MongoDB

Vous allez dans ce TD vous exercer à faire des requêtes à un serveur MongoDB. Nous aurons ainsi exploré toutes les étapes de la création du projet :

1. la modélisation initiale (séance 1)
2. la conception du front (HTML/CSS, JavaScript et React) (séances 2 à 5)
3. la mise en place du serveur d'API et des échanges client-serveur d'API (séance 6)
4. la définition précise des web services (séance 7)
5. aujourd'hui, les échanges entre le serveur d'API et le serveur de base de données

8.1 Installation et utilisation...

Nous pouvons soit utiliser un serveur distant, en utilisant le service Atlas [1], qui met à disposition une version gratuite avec quelques limitations, pour les expérimentations, ou la mise en place d'un serveur local.

8.1.1 ... sur votre propre machine

Le serveur MongoDB est téléchargeable depuis l'adresse suivante : urlhttps ://www.mongodb.com/try/download/community
Pour un serveur local, il y a deux composants à installer :

- le serveur lui-même :
  - Sous Linux : de préférence depuis le dépôt de paquets de votre distribution, sinon en téléchargeant le script d'installation. Il pourra être lancé soue la forme d'un démon.
  - Sous Windows, le plus propre est de l'installer sous la forme d'un service
- le *driver* pour Node.js, qui permet à Node de dialoguer avec le serveur. Il s'installe simplement avec `npm install mongodb`

Avant de pouvoir l'utiliser, il faudra lancer le serveur :

- sous Windows, en lançant en tant qu'administrateur l'application Services, puis sélectionnant le serveur MongoDB et démarrant le service. Par défaut, il est en démarrage automatique ; si vous ne comptez pas utiliser MongoDB au quotidien, passez-le en mode Démarrage manuel
- sous Mac et Linux, en tant que service (`sudo systemctl start mongodb.service` permanent ou non), ou pour une durée limitée à votre session en tapant `mongod` en ligne de commande.

8.1.2 ... sur les machines de la PPTI

MongoDB a été installé sur les machines de la salle, et le démon tourne. Vous pouvez utiliser `mongosh` en ligne de commande ou Compass (si le menu « Activités » ne le trouve pas, tapez en ligne de commande `mongodb-compass`).
Enfin, n'oubliez pas d'installer le *driver* pour Node pour votre projet : `npm install mongodb`.

8.2 MongoDB et Node.js

Nous allons ici définir le format de données pour nos messages.

8.2.1 Donnez un exemple de message à stocker (en JSON) pour votre site. Considérez pour le moment que la date n'est qu'une chaîne de caractères `"XXX"`.

1. https://www.mongodb.com/cloud/atlas/register

```
{
        author_id: 158,
        date: "XXX",
        text: "Voilà le texte du commentaire",
        id_forum=0
}
```

On ne s'intéresse pas aux dates ici.

8.2.2 Comment insérer un message en MongoDB avec NodeJS ?

En supposant qu'on utilise la collection `messages` :

```
const dbName="asso";

const newMessage = {
        author_id: "158",
        date: " XXX ",
        text: "Voilà le texte du commentaire"
}

const insertMessage = await client.db(dbName).collection("messages").insertOne(newMessage);
```

8.2.3 Comment afficher les messages ? Quel est le résultat obtenu ?

```
const cursorListMessages  = await client.db("asso").collection("messages").find();

// Pour afficher en console, il faut transformer le cursor en array
const arrayListMessages = await cursorListMessages.toArray();
console.log(arrayListMessages);
```

Résultat :

```
[
  {
    _id: new ObjectId("641f7400ce87cc393fb4ed2c"),
    author_id: '158',
    date: ' XXX ',
    text: 'Voilà le texte du commentaire'
  }
]
```

8.3 Requêtes

8.3.1 Mongo crée automatiquement un identifiant `_id`. Écrire une commande pour afficher le texte et l'`_id` du message

Vous consulterez la documentation de la méthode find() afin de spécifier parmi les options le fait de ne garder que les deux propriétés `text` et `_id` :

```
query = {...}
options = {
        projection : {...}
}
```

puis `(...).find(query, options);`

```
const query={};
const options={ projection : { text: 1, _id: 1} }
const cursorListMessages  = await client.db("asso").collection("messages").find(query,options);

const arrayListMessages = await cursorListMessages.toArray();
console.log(arrayListMessages);
```

Résultat :

```
[
        {
                _id: new ObjectId("641f7400ce87cc393fb4ed2c"),
                text: 'Voilà le texte du commentaire'
        }
]
```

8.3.2 Écrire le masque de requête permettant de connaître tous les messages écrits par un utilisateur ayant un `author_id` donné. Écrire le code permettant d'afficher le texte des messages de cet auteur.

```
const query={author_id: "159"};
const options={ projection : { text: 1} }
const cursorListMessages  = await client.db("asso").collection("messages").find(query,options);
```

8.3.3 Écrire la promesse `getMessageID(idAuthor, text)` qui renvoiel'identifiant du message écrit par un auteur donné et dont le texte est également en paramètre. Donner un exemple d'utilisation.

```
async function getMessageID (idAuthor, text) {
        const query = {author_id : idAuthor, text: text};
        const options = {projection: {text: 1}}

        const cursorListMessages  = await client.db("asso").collection("messages").find(query,options);
        const arrayListMessages = await cursorListMessages.toArray();

        return (arrayListMessages[0]._id)
}
```

Puis pour l'utiliser, comme c'est une promesse :

```
const IDres = await getMessageID("158", "Voilà le texte du commentaire");
```

8.3.4 Écrire le code permettant de retrouver les messages écrits par les utilisateurs 155 et 198.

```
(...).find({
        "author_id": {$in: [158,198]}
        }
)
```

8.3.5 Écrire le code qui permet d'afficher le message qui a l'identifiant récupéré par getMessageID.

```
const IDres = await getMessageID("158", "Voilà le texte du commentaire");

const mess = await client.db(dbName).collection("messages").findOne( {_id: IDres} );
console.log(mess);
```

8.4 Initialisation de la base de données

Il faudra vous assurer qu'une personne installant votre projet pour le tester puisse le faire avec des comptes et des messages déjà créés. Le but de cet exercice est vous montrer le principe de base pour initialiser la base de données avec des collections et document permettant de tester (et corriger !) votre projet.
Compléter le code du fichier `indexVide.js`. Vous allez compléter la promesse `fillDB` :

1. créer un objet `test` (peu importe son contenu en fait, il suffit qu'il soit non vide) :

   ```
   test={
           init: true
   }
   ```

2. connectez-vous à la collection nommée `colltest`de la base `dbName`
3. tester si un objet possédant une propriété `init` valant `true` existe dans la collection `colltest` : si c'est le cas, afficher dans la console le texte "Trouvé", sinon afficher le texte "Non trouvé" puis ajouter l'objet `test` dans la collection `colltest`

Normalement, à la première exécution de ce code, la console affiche "Non trouvé", et à partir de la deuxième, elle affiche le texte "Trouvé".
Dans le cadre de votre projet, vous créerez une base nommée d'après les initiales de votre groupe (par exemple, pour Maude Zarella et Agathe Zeblouse, "mzaz") et remplacerez les affichages dans la console par votre code permettant d'ajouter les documents utiles.

```
const {MongoClient} = require('mongodb');

async function fillDB(client,dbName){
                test={
                        init: true
                }
                const col1 = await client.db(dbName).collection("colltest");
                const testInit = await col1.findOne(
                        {
                                init: true
                        }
                );

                if (testInit) {
                        console.log("Trouvé");
                }
                else {
                        console.log("Non trouvé");
                        await col1.insertOne(test);
                }
}

async function main(){
        // URI de connexion. À modifier si on n'est pas en local
        const uri = "mongodb://localhost";
        const client = new MongoClient(uri);
```

```
        try {
                // Connexion au serveur
                await client.connect();

                // ici le code à exécuter...
                await fillDB(client,"gc");

        } catch (e) {
                // si une des promesses n'est pas réalisée
                console.error(e);
        } finally {
                // une fois que tout est terminé, on ferme la connexion
                await client.close();
        }
}
// main est une promesse, donc peut afficher un message en cas d'échec
main().catch(console.error);
```

TME

Mise en place de la base de données

Réfléchissez à l'organisation de votre base de données : quelles collections faut-il mettre en place ? Quels champs faut-il incorporer dans les documents ? avec quel type de données ?

Et la suite...

Toutes les pièces du puzzle que vous devez concevoir sont entre vos mains : composants React, serveur Express, échanges de données, base de données.
Il n'y aura *a priori* plus de sujet de TD et TME cadré ; c'est à vous de vous organiser afin de mener à bien le projet en profitant de la présence des enseignants en cas de souci. Le temps réservé pour les TD et TME chaque semaine ne sera pas suffisant, vous devrez travailler en autonomie.