TP4 - Intégration de la base de données MongoDB
27 octobre 2016

1. Objectifs

Le but de ce travail pratique, réalisé en équipe de deux personnes exactement, est d'intégrer une base de données à votre application serveur Express/NodeJS.

2. Introduction

Depuis le début du trimestre, vous implémentez un site web qui génère des questionnaires (quiz) pour les personnes qui désirent tester leurs connaissances sur les notions du Web. Nous nous attardons principalement sur des questions sur le domaine du HTML5, CSS3 et Javascript, mais nous pouvons étendre ces questions à d'autres technologies du Web telles que HTTP, NodeJS, AngularJS 2, etc.

3. Travail à réaliser

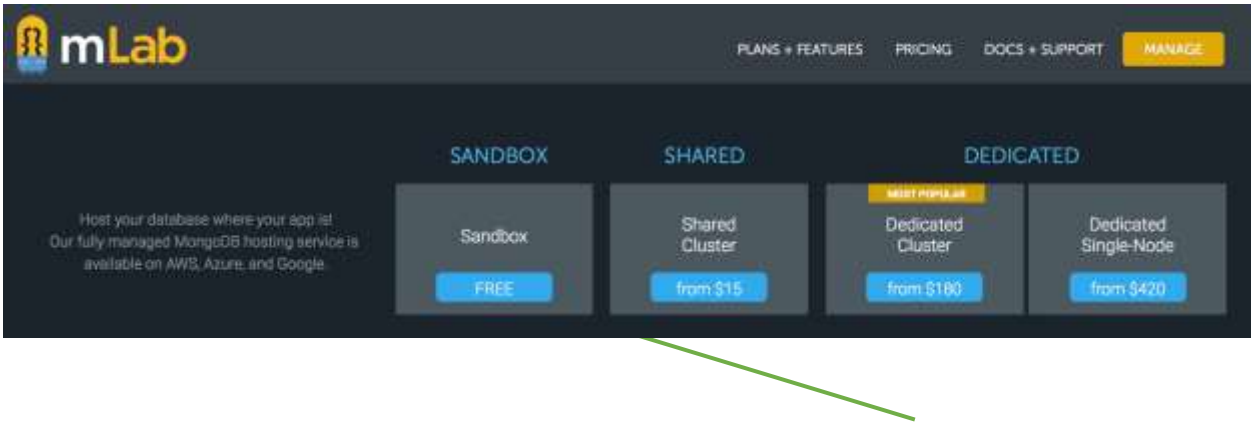
Au TP3, vous avez migré votre site web du TP3 vers une application serveur. Présentement, l'ensemble de vos questions se trouvent directement dans vos fichiers du serveur. De plus, vos réponses sont présentement validées du côté client et les scores sont stockés du côté client

Au cours du TP4, vous utiliserez l'API Mongoose pour communiquer avec une base de données MongoDB.

Les objectifs principaux de ce TP sont :

- Pouvoir soumettre des questions/réponses par l'interface web et stocker ces questions/réponses dynamiquement sur une base de données Mongo.
- Valider les réponses du côté serveur.
- Stocker les scores et le progrès des usagers sur une base de données Mongo (migration des local storage).
- Pouvoir récupérer les statistiques sur la base de données.

a. Vous allez créer votre base de données à distance dans le site www.mongolab.com



Pour des petites bases de données, c'est gratuit. Utilisez l'option **<<SANDBOX>>**.

Assurez-vous d'utiliser un mot de passe que vous pourrez partager avec votre coéquipier si nécessaire, pour éviter des complications.

b. Pour avoir un API sur les fonctionnalités de MongoDB, je vous conseille d'utiliser Mongoose. La librairie existe pour Node et peut être ajoutée aux dépendances de la même manière que les librairies étaient ajoutées aux TP précédents (soit avec **npm install –save**, soit en modifiant package.json manuellement (voir détails plus bas)). Référence de la librairie : http://docs.mlab.com

D'autres lectures recommandées pour structure votre code avec Mongoose:

- https://zellwk.com/blog/crud-express-mongodb/
- http://mongoosejs.com/docs/connections.html
- http://cwbuecheler.com/web/tutorials/2014/restful-web-app-node-express-mongodb/
- http://scotch.io/tutorials/javascript/build-a-restful-api-using-node-and-express-4
- http://mongoosejs.com/docs/api.html
- La librairie est populaire et des recherches Google seraient fructueuses aussi.

Pour extraire des questions aléatoires, je vous conseille d'utiliser la librairie *mongoose-simple-random*. Elle est très simple à utiliser.

Le fichier *package.json* contient l'ensemble des modules de dépendances de votre projet. Voici les dépendances à ajouter pour l'utilisation de Mongoose.

```
"dependencies": {
        ...
        "mongoose": "4.6.4",
        "mongoose-simple-random": "0.3.3"
}
```

Chaque fois que vous modifiez le fichier package.json, il faut lancer la commande *npm install* pour installer ou modifier les modules de dépendances.

Voici les directives de ce travail pratique :

1. Créer un fichier « ./lib/db.js » pour configurer Mongoose pour qu'il puisse communiquer avec votre base de données.
   Inclure ce fichier dans « app.js » avec « require('./lib/db'); »

   La structure du fichier db.js sera la suivante :

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

   var Todo = new Schema({
       user_id    : String,
       content    : String,
       updated_at : Date
   });

   mongoose.model( 'Todo', Todo );
   mongoose.connect( 'mongodb://localhost/express-todo' );
   ```

   (source : http://dreamerslab.com/blog/en/write-a-todo-list-with-express-and-mongodb/)

   Ce fichier va contenir le Schema de vos questions et les fonctions pour communiquer avec votre BD. Vous pourrez donc communiquer avec la base de données en travaillant sur des objets.
   Pour pouvoir faire appel à ces objets à partir de vos autres fonctions, vous pouvez vous référer aux exemples sur http://dreamerslab.com/blog/en/write-a-todo-list-with-express-and-mongodb/. La partie *Create items and save* présente des exemples.

2. Migrer l'ensemble des questions sur la base de données MongoDB..
   Pour ce faire, il faut créer une nouvelle page côté client (par exemple, « /ajouterQuestion ») qui contient un formulaire pour ajouter des questions à la BD MongoLab.

3. Pour peupler votre base de données la première fois, vous pouvez évidemment insérer les questions manuellement avec l'interface du site MongoLab, pour ne pas avoir à le faire une à la fois.

4. On doit retrouver un minimum de validation du côté client dans le formulaire pour ne pas soumettre n'importe quoi. Par exemple, il ne faudrait pas que la question soit vide, qu'il y ait moins de deux choix de réponses, que la bonne réponse soit en dehors de la portée des choix de réponse, etc.

5. Cette vérification doit être répétée du côté serveur. Si la soumission est invalide, le serveur devrait retourner une erreur 40x.

   Lorsque le formulaire est envoyé et que le format est valide, la question est enregistrée dans MongoDB. Utilisez la bonne réponse HTTP 2xx.

6. La soumission de la nouvelle question se fait de manière asynchrone (AJAX), avec une retroaction pour dire si elle a réussi ou pas.

7. Il faut respecter la syntaxe REST pour les requêtes du client vers le serveur. Donc, le chemin (URL) représente une ressource. Les verbes http représentent l'action à prendre : post pour créer une entrée, get pour lire une entrée, delete pour supprimer, put pour remplacer, patch pour modifier.

   Par exemple, **pour une autre application que la vôtre**, on aurait :
   Post sur www.example.com/users/ pour créer un nouvel usager, et obtenir l'information sur l'objet créé en réponse.
   Get sur www.example.com/users/ pour lire la liste de tous les usagers,
   Get sur www.example.com/users/12345 pour obtenir les détails de l'usager 12345,
   Delete sur www.example.com/users/12345 pour supprimer l'usager 12345
   Delete sur www.example.com/users/ pour supprimer tous les usagers.
   Post sur www.example.com/users/12345/creditCartds/ pour ajouter une nouvelle carte de crédit au compte de l'usager.
   Etc.

8. Modifier l'ensemble de vos fonctions du TP précédent pour pouvoir extraire les questions avec Mongoose à partir de MongoLab, plutôt qu'avec le fichier texte statique.

9. Les fonctionnalités du site web doivent être adaptées à un nombre inconnu de questions.

   Dans ce cas, vous devez modifier la validation de votre formulaire pour le nombre total de questions permises.

   Par exemple, supposons que j'ai 5 questions HTML, 3 questions CSS, et 1 question JS.

Dans ce cas, si l'utilisateur choisit le domaine HTML, alors le nombre de questions maximal permis est 5.
Donc, il faut communiquer avec la BD pour obtenir le nombre de questions pour un domaine.

10. Sur la page d'ajout de questions, ajouter une autre option pour vider toutes les questions de la BD Mongo.

11. Les statistiques courantes du test rapide ou de l'examen seront mises à jour selon les questions répondues par l'utilisateur, de la même manière que les TP précédents.

12. Le stockage des statistiques devra être migré sur la base de données plutôt qu'être en localstorage. Vous devrez créer un API REST similaire à l'interface des questions pour gérer ceci.

13. La vérification de la bonne réponse se fera du côté serveur. Ainsi, contrairement au TP3, la bonne réponse ne sera pas envoyée au client en même temps que la question. Ce n'est que lorsque le client soumet sa bonne réponse que le serveur retournerait si la réponse était bonne. Le résultat du client serait déjà sauvegardé dans la même opération et son progrès/statistiques seront mis à jour.

14. L'était de chaque client (son progrès dans l'examen) devra donc aussi être sauvegardé sur le serveur.

15. Vous n'avez pas à gérer l'accès des usagers (pas de mots de passe à gérer).

16. Le système peut être considéré comme étant mono-usager. Il n'est donc pas nécessaire de demander l'id de l'usager, ni de stocker les résultats par nom d'usager. En d'autres mots, il n'y a qu'un seul usager qui utilise l'application. ~~Vous devrez identifier votre usager au début de la session, sur la page de tableau de bord. Ceci peut être aussi simple que de lui demander un nom ou id d'usager. Ce nom peut être stocké en localstorage et utilisé dans les requêtes subséquentes. Si cette valeur n'est pas présente, l'usager peut être redirigé au tableau de bord pour le faire.~~

17. Comme le progrès de l'usager est sauvegardé du côté serveur, l'usager devrait pouvoir quitter et revenir plus tard pour continuer. Cette fonctionnalité devrait être fonctionnelle tant que le progrès n'est pas effacé.

18. Les fonctionnalités générales du site web doivent fonctionner telles que pour les autres TP.

4. Évaluation

Globalement, vous serez évalués sur l'utilisation de Mongoose avec votre application serveur. Plus précisément, le barème d'évaluation est le suivant :

Exigences	Points
À venir.	

L'évaluation se fera en lançant la commande *npm start* à partir du dossier qui contient app.js (ou autre selon le nom que vous lui aurez donné). À partir de cette page, je devrais être capable de consulter toutes les pages de votre site web.

De plus, j'utiliserai ma propre base de données pour tester votre application.

6. Directives pour la remise

1. Pour la remise, on demande de placer votre code Express/NodeJS dans un dossier compressé nommé : TP4_matricule1_matricule2. Vous avez le choix du type de compression. Vous devrez exclure le dossier node_modules.

2. Vous devrez également créer un fichier nommé temps.txt à l'intérieur du dossier de votre projet. Vous indiquerez le temps passé **au total** pour ce TP.

3. Le travail pratique doit être remis **avant** 23h55 le dimanche 6 novembre 2016 sur Moodle.

7. Pénalités

1. La politique pour les travaux pratiques remis en retard: -20% pour chaque jour de retard.

2. Les directives 1 et 2 de la remise: -5% si ces directives ne sont pas respectées.

Bonne chance!