PROJET
MEAN

Nous arrivons au terme du module consacré à l'étude de la stack MEAN. Après avoir passé individuellement en revu les technologies qu'elle réunit au travers d'exemples mettant en évidence différents aspects de leur fonctionnement, le temps est maintenant venu de rassembler toutes ces connaissances pour les appliquer à la réalisation d'un (sempiternel) blog. Cela pourrait sembler anodin tellement le sujet a été épuisé. Cependant, il reste un support pédagogique de premier choix car il présente l'avantage de mettre en jeu les principaux concepts et mécanicismes que l'on retrouve sous une forme ou sous une autre dans les applications qui agitent le web. Par ailleurs il est toujours intéressant de pouvoir comparer les technologies web entre elles. Pour la petite touche d'originalité, nous tenterons l'intégration UI du fameux material design de Google (https://getmdl.io/templates/blog/index.html)

Nous partirons ici sur une architecture classique de type client/serveur en nous appuyant sur la stack MEAN. Nous nous imposerons donc le javascript comme langage de développement coté serveur avec Node. Pour sa partie cliente, les développements seront réalisées sur une plateforme (navigateur) dont la seule contrainte est de supporter AngularJS.

I - Backend Node/Express/Mongoose

L'une des premières étapes lors de la réalisation d'un projet consiste à déterminer l'ensemble des données sur lequel notre application devra travailler. Nous allons donc commencer dans un premier temps par définir nos données métiers.
Notre blog, qui se veut minimaliste, se contentera de ne gérer qu'une liste d'utilisateurs et leur postes associés, éventuellement suivis de commentaires.
Il se dégage tout naturellement de ce qui vient d'être dit 2 modèles métiers : User et Post (les commentaires faisant parti intégrante des postes)

Un utilisateur possédera les attributs suivants :

- **username** : l'identifiant
- **password** : le mot de passe
- **profileImageURL** : un lien pointant vers l'URL ou se trouve l'image du profile.
- **updated**: la date de la dernière mise à jour
- **saved** : la date de création

et un poste

- **title**
- **imageURL**
- **content :** le texte
- **comments** : un tableau de commentaires
- **created** : la date de création de l'article
- **user** : l'auteur de l'article. L'ObjectId de l'utilisateur auquel il fait référence

chaque commentaire d'un poste devra être structuré de la façon suivante :

- **title**

- **content**
- **user** : l'auteur du commentaire. L'ObjectId de l'utilisateur auquel il fait référence
- **created**

Tous nos objets seront stoqués sous forme de documents dans une base de données orientée documents (MongoDB) pilotée par un ODM (Mongoose).

1. Créez, dans le dossier racine encore vide de votre projet, un sous dossier **models** qui contiendra les fichiers **user.js** et **post.js**. Ces fichiers devront décrire vos modèles User et Post à l'aide de Mongoose.

2. Une fois nos entités définies, le besoin de disposer de données concrètes se fait très vite ressentir. Nous aimerions en effet pouvoir suivre l'évolution de notre travail en ayant à tout moment la possibilité de visualiser l'aperçu d'un site chargé en contenu. Un script de génération de données aléatoires est alors le bienvenu. Plusieurs bibliothèques implémentées dans différents langages existent pour vous simplifier le travail (vous pouvez également trouver des services web qui s'en charge). Vous êtes libre de la démarche à suivre mais je vous recommande d'opter pour **faker** (https://www.npmjs.com/package/faker), bibliothèque javascript des plus populaires en la matière. Disponible dans le dépôt npm, il s'intègre parfaitement à notre environnement de travail.
Une fois lancé, le script devra se charger de **générer un ensemble d'utilisateurs et de postes fictifs ET** de **peupler la base de données** avec. Vous devrez donc combiner faker et mongoose au sein d'un même script que l'on appellera sobrement **populate.js** et qui se trouvera dans le sous-dossier **data** présent à la racine de votre projet. Ce script importera les modèles définis plus haut et devra instancier 4 utilisateurs et 20 postes associés aléatoirement aux 4 utilisateurs créés, avant des les sauvegarder en base. Le nombre de commentaires variera entre 0 et 5 et de la même façon que pour les postes, chaque commentaire devra se rapporter à un utilisateur existant.

Notre base de données est maintenant prête à être exploitée. Nous pouvons commencer à implémenter une **API HTTP CRUD** pour l'exposer et servir les documents qu'elle contient sur le web. Pour ce faire, nous allons structurer l'arborescence de notre projet de la façon suivante :
A la racine du projet :
Un fichier **server.js** contiendra le code du lancement de notre serveur web.
Un dossier **routes** contiendra un sous dossier **api** qui contiendra lui même 2 fichiers **user.js** et **post.js**. Ces fichiers définiront respectivement l'API HTTP des documents utilisateur et poste.
Un dossier **public** contiendra un unique fichier **index.html**.

3. Sans vous soucier de l'API HTTP pour le moment, rédigez le script **server.js** qui se chargera de servir le contenu du dossier public à la racine du site (le dossier public sera mappé vers l'URL http://localhost:3000/). Le seul middleware à intégrer ici est **express.static**.

Après avoir vérifié le bon fonctionnement du script en plaçant des fichiers dans de dossier public et en constatant l'affichage de leur contenu par le navigateur, nous allons pouvoir commencer à rédigez les fichiers routes/api/user.js et routes/api/post.js.

Il s'agira là de définir les APIs HTTP à proprement parler. Concentrons d'abord sur les postes, et nous appliqueront ensuite le même raisonnement pour les utilisateurs.

Voici les routes que l'on souhaite définir :
/api/posts en GET : récupère l'ensemble des postes
/api/posts en POST en transmettant un post : crée un poste
/api/posts en DELET en transmettant un id: supprime un poste
/api/post/:id en GET : récupère un poste
/api/post/:id en PUT : met à jour un poste

Puisque les APIs HTTP définissant des arborescences de routes biens distinctes, les scriptes qui en sont à l'origine peuvent être considérés comme autant de sous application express indépendantes. Ces sous applications autonomes, développées à l'aide de l'objet Router qui implémente l'interface des middlewares expresse, s'intègrent tout aussi simplement à une application expresse, rendant le code extrêmement modulaire.

4. Définissez dans le fichier routes/api/post.js un objet Router responsable de la gestion de l'API HTTP du modèle Post.
De la même manière, rédigez le script routes/api/user.js gérant l'API HTTP du modèle User.

Ces scripts feront appels
- aux fichiers où sont définis nos modèles,
- à la librairie mongoose pour réaliser les opérations CRUD en base de données
- à express qui fournira l'armature web.

Validez votre API HTTP à laide d'un générateur de requête HTTP (postman, curl si vous êtes un adepte de la ligne de commande, etc …)

4. bonus : factorisez le code en unifiant la gestion des APIs HTTP User et Post dans un seul et même fichier api.js.

Voilà, nous en avons terminé avec le backend ... ou presque. Il reste encore un aspect à couvrir et non des moindres :
- Notre application devra être capable d'authentifier les utilisateurs pour leur attribuer les contenus qu'ils rédigent.
- Devra protéger notre API HTTP en mettant en place un contrôle d'accès.

Cette couche de sécurité vous est fournie. Elle a été réalisée à l'aide des JWT (pour json web token), technique offrant un niveau de fiabilité supérieur à celle des sessions. Tel que défini dans le fichier **login.js**, la politique de sécurité mis en place limite la visibilité des informations qu'aux utilisateurs authentifiés. Nous pourrons revenir et modifier ce comportement par la suite si nous le souhaitons.

5. Intégrez la couche d'authentification au projet. Pour cela placerez ce fichier dans le dossier des **routes** (mais à l'extérieur du dossier **api**) et renseignez votre application de l'existence de ce nouveau Router. Vous devrez également effectuer l'opération d'authentification dans les routes que vous introduisez.