POLYTECH ANNECY-CHAMBÉRY UNIVERSITÉ SAVOIE MONT BLANC

Data 834

TP 1 (Redis)

Vous venez d'être recruté pour faire une mission auprès d'une startup qui propose ses données sous la forme d'un *Software as a Service*. La startup commence à être victime de son succès avec une utilisation de plus en plus soutenue de ses services. Il est nécessaire de remédier à ça en limitant le nombre d'appels aux services de l'entreprise (et en monétisant pour une utilisation plus intensive).

Vous allez pour cela devoir consigner le nombre d'appels par utilisateur enregistré et connecté. La startup considère qu'il est possible de requêter leurs services à raison de 10 appels par fenêtre de 10 minutes.

Les outils à utiliser sont Node.js (pour le serveur REST), MongoDB (pour la gestion des utilisateurs), et Redis (pour la gestion du nombre d'appels par utilisateurs).

Suivons un ensemble d'étapes pour la réalisation de ce projet.

1. Assurez-vous que vous disposez de MongoDB, Node.js et Redis sur votre machine et qu'ils fonctionnent. Il faut bien entendu, à ce moment-là que MongoDB et Redis soient lancés. Pour MongoDB, mongod --dbpath ./data et pour Redis redis-server (approche différente sur Mac)
2. Créez un dépôt Git sur Github pour le projet
3. Clonez votre projet sur votre machine
4. Dans le répertoire du projet, créez un projet Node.js (avec npm init), à part le nom du développeur, pas forcément besoin de remplir le reste
5. L'approche va ressembler fortement à ce que nous avons fait sur Info 734, créez d'abord un serveur minimal puis venez modifier les routes afin qu'elles n'aient que les routes correspondant à l'API. Comme ici, l'objectif est avant tout de réaliser la partie Utilisateur et Redis, ne vous cassez pas la tête sur les routes pour l'API, une seule qui renvoie de la fausse donnée suffira amplement
6. Nous avons donc maintenant un serveur REST que vous pouvez interroger avec Postman. Comme vous pouvez le voir, il n'y a aucun contrôle sur le nombre d'appels au risque d'écrouler le serveur. Il faut donc maintenant travailler sur la gestion des utilisateurs
7. On commence par créer le modèle pour l'Utilisateur, vous vous rappelez, on crée un répertoire dans le projet pour Utilisateur. Le modèle contiendra le nom, le prénom, l'email et le mot de passe. Comme bien entendu, nous faisons du travail professionnel,

nous cryptons le mot de passe. L'ensemble des instructions pour cela se trouve à : http://devsmash.com/blog/password-authentication-with-mongoose-and-bcrypt

8. Il faut maintenant ajouter les fonctions CRUD pour l'utilisateur (principalement d'ailleurs, C). Vérifiez leur fonctionnement avec Postman.
9. D'un côté, nous avons donc l'utilisateur et de l'autre l'API. Il faut donc relier les deux en s'assurant que seuls les utilisateurs enregistrés et connectés peuvent y accéder.
10. Créez donc la route pour que l'utilisateur se connecte. Il serait tout à fait possible que l'utilisateur passe à chaque appel son email et son mot de passe, mais ce n'est pas très sécurisé. Il est préférable d'utiliser des JsonWebToken. L'utilisation des tokens est assez simple et est résumée à : https://dev.to/_marcba/secure-your-node-js-application-with-json-web-token-4d4e
11. A chaque fois que l'utilisateur se connecte, il reçoit un token puis à chaque fois qu'il appelle le service, il passe le token pour s'authentifier (c'est par l'intermédiaire de Bearer dans Postman)
12. Il faudrait ensuite s'assurer que seuls les utilisateurs connectés peuvent accéder aux services (par l'intermédiaire de passport-local) mais nous pouvons nous garder ça pour la fin.
13. Nous pouvons enfin utiliser Redis et stocker un compteur associé à chaque token. Créez donc les fonctions nécessaires pour obtenir la valeur stockée et l'incrémenter. (avec https://github.com/NodeRedis/node-redis)
14. Finalement, ajoutez ces fonctions dans les appels aux services et veillez à contrôler le nombre d'appels possibles.

Testez par l'intermédiaire de redis-cli et de votre serveur que tout fonctionne correctement.
Les améliorations possibles à partir de là : plus de fonctions sur la gestion des utilisateurs et la possibilité de monétiser avec Stripe.