1 RESTful APIs sous Express avec JWT auths

1.1 Création d'une RESTfull API protégée par JWT pour myMoovies

Veuillez continuer le développement de **myMoovies**. Vous allez créer une nouvelle version de la RESTful API de **myMoovies**, sous Express, afin de mettre à disposition l'authentification d'utilisateurs et de permettre de protéger toutes les opérations de mises à jour de films.
NB : Dans cet exercice, les données ne sont pas réellement protégées. Cela sera fait à l'exercice prochain.

Afin de réaliser cet exercice, voici les contraintes d'implémentation :

- Veuillez ajouter les ressources permettant d'authentifier un utilisateur et de générer un token :

URI	Méthode	Auths?	Fonction
auths/login	**POST**	Non	Vérifier les « credentials » d'un User et renvoyer le User et un token JWT s'ils sont OK `{ "username": "charly", "token": "eyJhbGciOiJIU…" }`
auths/register	**POST**	Non	Créer une ressource User et un token JWT et les renvoyer `{ "username": "sam", "token": "eyJXxx…" }`

- Pour ajouter les ressources de type **users**, utilisez les fichiers offerts par la démo js-demos/backend-restful-api/restful-api-essentials/step5/ . Cette démo offre *:*
  - La persistance des ressources de type **users**: les données associées aux utilisateurs sont sauvées dans un fichier **.json**
  - les passwords « en clair » dans le fichier **.json ;** pour cet exercice, on ne souhaite pas encore protéger les données des utilisateurs.
  - Un code structuré : le fichier **/model/users.js** propose un « Fat Model » contenant toute la logique d'accès aux ressources de type **users** dans.

- Veuillez mettre à jour la RESTful API pour la collection de films afin que les opérations de mises à jour de données (Création, Update & Delete) soient protégées par une autorisation JWT : seul un utilisateur authentifié doit pouvoir réaliser ces opérations. Voici les opérations de l'API :

URI	Méthode	Auths?	Fonction
films	**GET**	Non	READ ALL : Lire toutes les ressources de la collection
films?minimum-duration =value	**GET**	Non	READ ALL FILTERED : Lire toutes les ressources de la collection selon le filtre donné
films/{id}	**GET**	Non	READ ONE : Lire la ressource identifiée
films	**POST**	JWT	CREATE ONE : Créer une ressource basée sur les données de la requête
films/{id}	**DELETE**	JWT	DELETE ONE : Effacer la ressource identifiée
films/{id}	**PUT**	JWT	UPDATE ONE : Remplacer l'entièreté de la ressource par les données de la requête

- Assurez-vous que vos fichiers de données **.json** ne soient pas tracké par Git : ceux-ci ne doivent pas être repris dans le web repository.
- Veuillez tester toutes les fonctions de la RESTful API pour les collections de **auths** et de **films** à l'aide du **REST Client** dans VS Code (extension à installer au sein de VS Code). Veuillez ajouter vos requêtes au sein du fichier **auths.http** et mettre à jour les requêtes de **films.http** (pour l'autorisation JWT) dans le répertoire **RESTClient** du dossier associé à cet exercice.

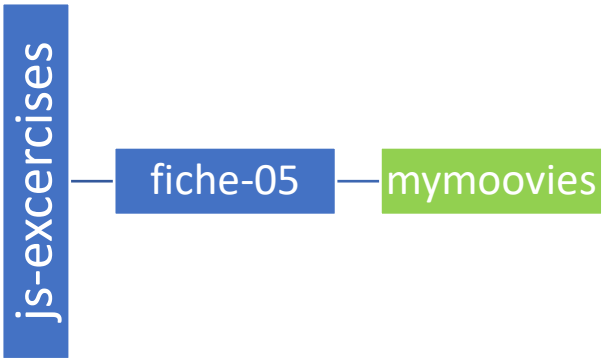
- *N'hésitez pas à repartir de la version de l'API que vous avez développé à la fiche 04. Pour ce faire, copier/coller de **/js-exercises/fiche-04/mymoovies** vers **/js-exercises/fiche-05/mymoovies***

Exercices

- *Besoin d'inspiration pour créer votre API ?*
  N'hésitez pas à lire le fichier ***README.md*** *et à copier/coller les fichiers utiles qui sont offerts dans la démo « Création d'une RESTfull API pour une pizzeria : Step 5 – Refactor : création d'un middleware d'autorisation JWT » : [js-demos/backend-restful-api/restful-api-essentials/step5/]*
- *N'oubliez pas d'installer le package pour gérer des tokens JWT :* ***jsonwebtoken***
- *Dans la démo Step 5 dont le lien est donné ci-dessus, il existe un middleware d'autorisation des accès aux ressources. Utilisez le (****/utils/authorize.js****) afin de sécuriser les opérations de type « Create », « Update » et « Delete » sur les ressources de type film.*
- *Pour tester que les opérations sont bien protégées par JWT, faites plusieurs requêtes via REST Client d'ajout d'un film : une requête sans* ***Authorization*** *header, une requête avec un* ***Authorization*** *header mais sans avoir de token, une requête avec un* ***Authorization*** *header contenant le token de l'utilisateur manager… Inspirez vous des requêtes faites dans le Step 5, fichier* ***/RESTClient/pizzas.http*** *et* ***/RESTClient/auths.http****.*

Une fois votre application terminée, veuillez faire un **commit** de votre code avec le message suivant : « myMoovies : step 5 : RESTful API. New : JWT auths : unsecure data ».

Le code de votre application doit se retrouver dans ce dossier (en vert) de votre repository local et de votre web repository (**js-excercises**).



2 Sécurisation d'une RESTful APIs

2.1 Sécurisation de la RESTfull API pour myMoovies

Vous allez créer une nouvelle version de la RESTful API de **myMoovies**, afin de :

- sécuriser les données des utilisateurs ; pour ce faire, leur passwords doit être hachés ;
- pour les ressources de type **films**, veuillez échapper les caractères dangereux qui pourraient amener à des attaques XSS ;
- Veuillez ajouter un rôle aux utilisateurs afin d'autoriser l'accès à des opérations seulement aux utilisateurs ayant le rôle approprié ;
- Veuillez ajouter l'opération de lecture de tous les utilisateurs, seulement pour les utilisateurs authentifiés dont le rôle est « admin ».
- Veuillez ajouter l'opération de mise à jour d'un utilisateur seulement pour l'utilisateur lui-même ; la mise à jour du username et du password de l'utilisateur ne doit pas être possible via cette opération.
- En résumé, voici les opérations attendues sur les **users** :

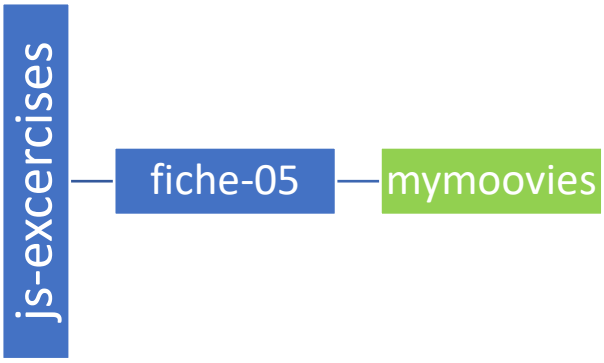
URI	Méthode	Auths?	Fonction
users	**GET**	JWT	READ ALL : Lire toutes les ressources de la collection seulement si l'utilisateur a le rôle d'admin
users/{username}	**PUT**	JWT	UPDATE ONE : Remplacer la ressource par les données de la requête

URI	Méthode	Auths?	Fonction
			seulement si c'est l'utilisateur même qui tente de modifier ses données NB : le username et le password ne doivent pas pouvoir être mis à jour ⇨ C'est juste le rôle actuellement qui peut être mis à jour.

Une fois votre application terminée, veuillez faire un **commit** de votre code avec le message suivant : « myMoovies : step 6 : update : RESTful API secured & roles. New : update a user, read all users ».

- *N'hésitez pas à continuer le développement de l'exercice précédent. Git permettra de passer d'un exercice à l'autre grâce à vos commits.*
- *Besoin d'inspiration pour créer votre API ? N'hésitez pas à lire le fichier **README.md** et à copier/coller les morceaux de code utiles qui sont offerts dans la démo « Création d'une RESTfull API pour une pizzeria : Sécurisation de RESTful APIs : Hachage d'information et protection contre les attaques XSS » : [js-demos/backend-restful-api/restful-api-essentials/step6/]*
- *Pensez à installer les nouveaux packages dont vous avez besoin :*
  - *pour hasher les passwords : **bcrypt***
  - *pour « échapper » les caractères dangereux : **escape-html***
- *Certaines fonctions de votre modèle s'exécutent en asynchrone, notamment **register** et **login**. Pensez à faire comme dans le Step 6, à ajouter des **async** / **await** au sein de **/route/auths.js***
- *Pour la gestion des ressources de type **users**, pensez à ajouter un router pour les traiter.*

Le code de votre application doit se retrouver dans ce dossier (en vert) de votre repository local et de votre web repository (**js-excercises**).



3 Exercice optionnel

3.1 RESTful APIs sous Express avec JWT auths

3.1.1 RESTful API pour un headless CMS

S'il vous reste du temps, n'hésitez pas à développer une nouvelle RESTful API qui vous serait utile, soit à développer ce qui vous est proposé dans cet exercice optionnel.

Peut-être avez-vous déjà entendu parler de cette mode montante dans le Web qu'est la JAMstack ?

Nous souhaitons développer un Headless CMS, c'est-à-dire un outil permettant de créer du contenu, très souvent utilisé dans le cadre de sites JAMstack.

Un Headless CMS fournira une interface web pour créer du contenu (ou des données), ainsi qu'une RESTful API mettant à disposition ce contenu.

L'Headless CMS que nous souhaitons développer doit permettre de créer le contenu associé à un blog.
Un blog sera structuré en pages. Une page aura un id, un titre, un URI, un contenu, un auteur ainsi qu'un statut de publication.
Le statut de publication d'une page peut actuellement prendre comme valeur :

- « hidden »
- « published »

Seul un utilisateur authentifié pourra créer des pages.

La modification d'une page, tout comme la suppression d'une page ne pourra se faire que par son auteur.
L'ajout ou la modification d'une page ne sera possible que si le statut de publication donné correspond à un des deux statuts actuels (« hidden ou « published »). Sinon un message d'erreur devra être renvoyé.
La lecture d'une page, ou de toutes les pages, pourra être réalisée par n'importe quel utilisateur authentifié pour les pages dont le statut vaut « published ».
Pour les pages dont le statut vaut « hidden », seul leur auteur pourra les lire.
A la lecture d'une ou plusieurs page, on veillera à ce que les données associées à l'auteur d'une page ne contiennent pas le password.

Veuillez développer une RESTful API mettant à disposition les opérations décrites ci-dessus.
Dans un premier temps, veuillez formaliser les opérations associées à vos RESTful APIs sous forme d'un tableau dans un fichier **README.md**. Votre tableau devrait reprendre les conventions REST présentées dans les slides associées aux cours.
NB : ces conventions sont visibles dans les tableaux repris aux deux exercices précédents.

Comment créer un tableau dans un fichier Markdown ? C'est simplement un tableau HTML (**<table>**).

Lors de l'implémentation de vos RESTful APIs, veuillez tester toutes les méthodes offertes par votre RESTful API à l'aide de REST Client et de scripts associés.

Le code de votre application pourrait se retrouver dans ce dossier (en vert) de votre repository local et de votre web repository (**js-excercises**).