Spring API - TP 02

01 . La sécurité, c'est mon métier

L'objectif de ce TP sera de mettre en place le système de sécurité de notre API. La méthode présenté ici sera celle d'une Basic Auth (c'est à dire Login + Password). Si vous vous sentez l'âme d'un explorateur du code, vous pouvez implémentez une version avec JWT.

Actuellement, nous n'avons qu'une seule route. De plus, celle-ci ne mérite pas vraiment d'être protégée. Mais elle vous permettra d'expérimenter les différents mode (Admin/notAdmin/notAuth).

02 . La configuration

Comme toujours avec un projet Spring, une étape de gestion des dépendances est nécessaire. Petit rappel de ce que l'on doit ajouter dans le `pom.xml` :

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Copy

Ceci nous permettra d'utiliser Mongo et la gestion de sécurité ! Petite synchro et on est parti !

03 . Reprise (de volée)

Pour ce sujet, je vais considérer que vous êtes capable de reprendre les bases du sujet A4. Nous allons réaliser le supplément qui permet de hasher notre mot de passe avec Bcrypt.

📦 pkdxapi
├ 📂 configuration
| └ 📜 SecurityConfiguration.java
├ 📂 controllers
| ├ 📜 ApiExceptionHandler.java
| └ 📜 PkmnController.java
├ 📂 errors
| ├ 📜 APIException.java
| └ 📜 UserAlreadyExistException.java
├ 📂 models
| ├ 📜 PkmnType.java
| ├ 📜 UserData.java
├ 📂 repositories
| └ 📜 UserRepository.java
├ 📂 services
| ├ 📜 PkmnService.java
| ├ 📜 UserDataService.java
| └ 📜 CustomUserDetailsService.java
└ 📜 PkdxapiApplication.java

Voila l'architecture que vous devriez avoir (Vous pourriez avoir changé certains noms). Si certains points vous sembles obscure, n'hésitez pas à demander.

3.1 Passage au hasher

Pour cela, rien de vraiment compliqué, il nous faudra juste définir un bean au niveau de notre `SecurityConfiguration`.

```
@Bean
public static PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
}
```

Avec ceci, si vous relancé, vous ne devriez plus réussir à vous connecter à votre API. Vous aurez un message d'erreurs spécifiant que votre mot de passe ne semble pas Bcrypt friendly

Pour cela, premièrement vous pourrez virer toute vos documents dans votre BD mongo, les passwords ne sont plus valides.

Et deuxièmement, dans `CustomUserDetailsService`, si vous vous rappelez, vous avez intégré la notion de `{noop}` pour dire que le mot de passe était en clair. Il vous faudra le retirer !

3.2 Register machine

Pour le moment, il fallait directement donner l'instruction à la BD de créer un utilisateur. Cela sera bientôt de l'histoire ancienne ! A nous de créer une route POST `/users/register`.

Celle-ci aura en paramètre le login et le password (à vous d'établir le moyen de passer les données).

Cela passera donc par la création d'un :

- UserController (controllers)
- UserDataService (services)
- UserDTO (models)

Nous avons déjà le répository.

Le `UserController` va récupérer les données entrante pour créer un `UserDTO`. Puis, celui-ci sera passé au `UserDataService`. Ce dernier va instancier un `UserData` puis demander au `UserRepository` de le save. Puis le `UserController` va retourner un message de succès bien formaté avec un code 200 si tout s'est bien passé

3.3 Prouve que tu existe !

Une petite vérification si l'utilisateur existe déjà, puis lever une exception si c'est le cas, serait nécessaire. Car sinon Mongo réécrit sur le document existant.

Pour cela, rien de compliqué, nous allons ajouter une fonction `findByUsername(String username)` à notre repository. Pas besoin de définir la fonction (en même temps, c'est

une interface ^^), il suffit d'utiliser l'annotation @Query (cf : A2).

```
Vérifiez bien dans votre BD le nom du champs. Ce n'est pas forcement celui
donnée en java
```

Puis dans le service, nous allons créer une fonction qui retournera si oui ou non l'utilisateur existe :

```
private boolean usernameExist(String username){
//TODO retourner si un utilisateur à été trouvé
}
```

Pour l'exception, je vous invite à créer une classe `UserAlreadyExistException` qui héritera de `APIException`. Celle-ci ne fera qu'appeler le constructeur de son parent en passant le message et le HTTPStatus 409 ou 422 (CONFLICT ou UNPROCESSABLE_ENTITY)

04 . Toi tu vie, toi tu vie, toi tu ...

Il ne reste plus qu'a définir les différents droits des différentes routes. Toute ce passera donc dans notre `SecurityConfiguration` et plus précisément dans la fonction `securityFilterChain()`

Voici des exemples de configs (les routes ne sont pas forcement liés au projet) que vous pouvez essayer pour vérifier que la sécurité fonctionne :

```
.requestMatchers("/test").permitAll()
.requestMatchers(HttpMethod.POST, "/entity/**").hasAuthority("ROLE_ADMIN")
.requestMatchers("/users/**").permitAll()
.anyRequest().authenticated()
```

Vous pouvez créer une route `/users/test` protégé en admin par exemple.

A la fin, disons que notre route `/users/register` doit être accessible à tous tant dis que `/pkmn/types` nécessite d'avoir un compte (mais pas forcement admin)

05 . Un login un peu spécial

Dans ce TP, nous devons authentifier une majorité de nos routes avec une connexion login:password. Hors c'est aussi ce système que vous utiliserez certainement dans votre front. Comme nous n'avons pas mis en place de Token spécifique à récupérer, il nous faudra une route login un peu spécial.

Un peu de théorie, lorsque nous construisons notre requête HTTP, nous pouvons envoyer nos infos de connexion dans le header via la clé Authorization :

```
"Authorization" : "Basic bG9naW46cGFzc3dvcmQ"
```

Cette chaîne étrange est la version encodé en base64 de nos infos login:password. C'est donc cette info la qu'il faudra conservé dans votre Front (ceci n'est PAS sécurisé).

Côté back, il suffira d'avoir une route `/users/login` protégé par notre sécurité classique. Puis de renvoyer un code 200 (ou 204 vu que nous ne renvoyons pas de données). Si l'auth plante, le module de sécurité renverra une erreur