Spring API - A4 - Sécurité

Commencez par reprendre l'application Spring que nous avons réalisée petit à petit dans les précédents sujets. N'oubliez pas d'exécuter MongoDB pour avoir une base de données active.

Nous allons rajouter la notion d'authentification en utilisant des techniques intégrées à Spring.

01 . Authentification simple

Il faut commencer par activer la partie « sécurité et authentification » de Spring. Pour cela, éditez le fichier pom.xml et rajoutez, dans le nœud « dependencies » le code suivant :

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

Acceptez la synchronisation du projet. Exécutez votre serveur.

Testez à présent n'importe quel endpoint : vous devriez avoir à chaque fois la même réponse : **401 non autorisé** !

Observez la sortie console lors de l'exécution, vous devez y trouver le mot de passe autogénéré par Spring (qui est normalement différent du mien, bien entendu) :

```
Using generated security password: 89a9b3fa-484d-48f9-8234-9285d33d65f5
This generated password is for development use only. Your security
configuration must be updated before running your application in production.
```

Évidemment, la sécurité n'est pas encore totalement pertinente (comme le message l'indique) mais c'est une première étape.

Afin d'avoir l'autorisation d'accéder aux endpoints, il va falloir rajouter dans chaque requête le mot de passe, encodé dans l'entête.

Souvent situé dans l'onget Auth de votre client API, ici il faudra utiliser une authentification basic avec l'utilisateur : user et le mot de passe décrit précédemment.

02 . Utiliser un mot de passe personnalisé

La sécurité précédente est pas mal, mais évidemment insuffisante : le mot de passe est passé en clair dans l'entête, et il surtout il faut le déterminer une fois le serveur exécuté…

Il serait plus pertinent d'avoir un mot de passe personnalisé.

Il faut pour cela modifier la configuration par défaut de la sécurité, ce qui se fait simplement en créant une classe avec une annotation ad hoc.

Créez un dossier `configuration` au même niveau que les dossiers controllers, errors, models, repositories, services.

Créez dans ce dossier une classe `SecurityConfiguration` contenant le code ci-dessous :

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
                http.authorizeHttpRequests((authorizeRequests) ->
authorizeRequests
                                    .anyRequest().authenticated()
                        )    // authorize all http requests with
authentication
                        .httpBasic(Customizer.withDefaults()).csrf(csrf-
>csrf.disable()) ;  // disable csrf security to authorize post, patch &
delete
                return http.build();
    }
}
```

Relancez pour tester. Attention : le mot de passe autogénéré n'est pas le même qu'à l'exécution précédente. Vous devriez avoir le même fonctionnement que précédemment.

Nous allons à présent utiliser un autre mot de passe. Rajoutez dans `application.properties` un mot de passe (au choix), et dans la classe `SecurityConfiguration` l'opération suivante :

```
@Bean
public UserDetailsService userDetailsService(@Value("${password}") String
pass) {
        String password="{noop}"+ pass;
        return (username) -> new User(username, password,
AuthorityUtils.createAuthorityList("ROLE_USER"));

}
```

```
Note : {noop} indique que le mot de passe est passé en texte simple, qu'il
n'y aucune opération de chiffrage dessus. Si l'API est accédée en https,
cela ne pose aucun souci.
```

Relancez. Testez que les API ne fonctionnent bien qu'avec le mot de passe choisi. C'est mieux, mais pour l'instant tous les utilisateurs ont le même mot de passe, stocké en clair dans un fichier texte…

03 . Permettre plusieurs utilisateurs

3.1 Un nouveau modèle

Si nous souhaitons stocker dans notre base de données les utilisateurs, il va nous falloir un nouveau modèle pour gérer un nouveau document MongoDB…

Créez dans le dossier `models` une classe `UserData` annotée par @Document(« User ») et @TypeAlias(« UserData »). Cette classe comportera les propriétés (en lecture seule) suivantes :

- Login de type String (n'oubliez pas l'annotation @Id sur cet attribut)
- Password de type String
- IsAdmin de type booléen

Un constructeur par défaut doit être fourni, ainsi qu'un constructeur prenant en paramètre Login, Password et IsAdmin.

3.2 Un nouveau dépot

Créez dans le dossier 📁`repositories` une interface `UserRepository` qui hérite de MongoRepository<UserData,String>.

3.3 Un nouveau service

Créez dans le dossier 📁`services` une classe `CustomUserDetailsService` qui implémente l'interface `UserDetailsService`. Ne pas annoter cette classe !

Ajoutez un attribut de type `UserRepository`, initialisé par une injection de dépendances dans le constructeur.

Ajoutez un attribut de type `List<GrantedAuthority>` et initialisez-le dans le constructeur :

```
userAuthorities = AuthorityUtils.createAuthorityList("ROLE_USER");
```

Dans le code de l'opération `loadUserByName`, utilisez le dépôt pour retrouver l'objet `UserData` possédant le login username. Si cet objet n'existe pas, levez l'exception `UsernameNotFoundException`. Créez une instance de `org.springframework.security.core.userdetails.User` en utilisant l'objet lu :

```
new User(user.getLogin(), "{noop}"+user.getPassword(),userAuthorities)
```

3.4 Modification de la configuration de sécurité

Modifiez la classe `SecurityConfiguration`, et remplacez l'opération `userDetailsService` :

```
@Bean
public UserDetailsService userDetailsService(){
        return new CustomUserDetailsService(userRepository);
}
```

Vous aurez besoin d'ajouter un attribut de type `UserRepository` et une injection de

dépendances via le constructeur…

3.5 Test complet

Créez (avec l'extension VSCode mongoDb) un utilisateur dans la base (choisir un MDP simple pour le test ^^). Il faut pour cela insérer dans un document « User » :

```
use('test');
db.getCollection('User').insertOne(
        {
                "_id":  "toto",
                "password": "pa55",
                "admin": false,
                "_class": "UserData"
        }
);
```

Ajoutez 2 ou 3 utilisateurs, sur ce modèle.

Lancez l'application : vous devez normalement pouvoir accéder aux différentes API avec les utilisateurs que vous avez créés.

04 . Différents rôles/droits

Si nous souhaitons avoir une API qui n'est accessible qu'à certains utilisateurs, il faut définir différents rôles et indiquer à la configuration de la sécurité qui peut accéder à quoi…

Par exemple, si nous voulons limiter l'API /whois aux administrateurs…

Commencez par ajouter dans votre base MongoDB un utilisateur administrateur (champ admin = true).

Dans UserService, rajoutez un attribut :

```
private List<GrantedAuthority> adminAuthorities;
```

Initialisez-le dans le constructeur :

```
adminAuthorities =
AuthorityUtils.createAuthorityList("ROLE_ADMIN","ROLE_USER");
```

Modifiez loadUserByUsername pour avoir la bonne liste suivant si l'utilisateur chargé est un administrateur ou non.

Il ne reste plus qu'à modifier la chaîne de filtres de sécurité pour qu'elle devienne :

```
http.authorizeHttpRequests((authorizeRequests) -> authorizeRequests
        .requestMatchers("/person/**").authenticated()
        .requestMatchers("/whois").hasAuthority("ROLE_ADMIN")
)
.httpBasic(Customizer.withDefaults()).csrf(csrf->csrf.disable()) ;
```

Testez : normalement, seul un administrateur devrait pouvoir accéder à /whois, à présent !

Il est même possible de permettre un accès à une API à tout le monde, même non authentifié.

Rajoutez au filtre, par exemple, le code suivant pour qu'il ne soit plus nécessite de s'authentifier pour accéder à /hello :

```
.requestMatchers("/hello").permitAll()
```

05 . Aller plus loin

Les mots de passe sont stockés en clair dans la base de données… ce n'est pas idéal : il faudrait modifier cet état, en utilisant bien entendu les fonctionnalités de chiffrement fournies par Spring…

Voir Login and Registration REST API using Spring Boot, Spring Security, Hibernate and MySQL Database ou Password Encoding with Spring | Baeldung pour plus d'informations…

Une majorité d'API implémente une authentification à l'aide d'un JWT (JSON Web Token). Voici des ressources sur comment l'implémenter (en anglais) :

- Medium : JWT in Spring boot
- Spring Security With JWT for REST API