IDV-WEB4 / API REST / Etape 3 (Java)

Modalités

Type	Description
Dépôt	https://rendu-git.etna-alternance.net/module-9438/activity-51077/group-1010680
Dossier	Etape_3/
Correction	Moulinette
Taille de groupe	Groupe de 2

Objectifs

Notion	Description
SpringBoot	Sécuriser une application SpringBoot grâce aux tokens JWT

Consignes

Avant de commencer, vous devez copier votre projet de l'étape 2 dans un dossier nommé `Etape_3`.

1) Ajout des dépendances de sécurité

Vous devez modifier votre fichier `pom.xml` pour ajouter les dépendances suivantes :

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

<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>

<dependency>
```

```
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
  </dependency>

  <dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
  </dependency>
```

2) Mise en place de la sécurisation JWT

a) Création du package et installation du JwtTokenUtil

Ajoutez un package nommé `com.quest.etna.config`, c'est ici que nous allons stocker toute la configuration pour les tokens JWT.

Commencez en y copiant la classe `JwtTokenUtil` fournie avec ce sujet. Prenez le temps de la lire pour comprendre comment s'en servir au sein du projet.

Attention: La classe ne doit pas être modifiée, si le fichier comporte une erreur, tentez de télécharger d'une autre façon.

b) Création du modèle JwtUserDetails

En repartant de votre modèle `UserDetails`, vous allez créer le modèle `JwtUserDetails` qui doit implémenter l'interface `org.springframework.security.core.userdetails.UserDetails`. Le seul constructeur sera `JwtUserDetails(User user)`. Pensez à adapter chaque fonction en vous basant sur la variable `user`.

c) Création du service JwtUserDetailsService

Dans le package `config`, vous devez créer un nouveau service `JwtUserDetailsService` qui doit implémenter `UserDetailsService`. Vous aurez une seule fonction à compléter `loadUserByUsername` qui renverra un objet `JwtUserDetails` si l'utilisateur a été trouvé en base de données.

d) Modification du AuthenticationController

Dans le contrôleur `AuthenticationController`, ajoutez une nouvelle fonction nommée `authenticate` (POST /authenticate), qui prendra en entrée le username et le mot de passe de l'utilisateur. Cela connectera l'utilisateur trouvé et retournera un objet JSON contenant le token.
Pour connecter l'utilisateur vous devez utiliser la méthode `authenticate` de la classe `AuthenticationManager` et `UsernamePasswordAuthenticationToken`.
Pour générer le token vous devrez utiliser la classe `JwtTokenUtil`, attention aux paramètres en entrée, il y a un lien avec ce que vous venez d'implémenter.

e) Création du filtre

Dans le package `config`, ajoutez une nouvelle classe `JwtRequestFilter`. C'est un composant qui étend `OncePerRequestFilter`. Vous devez y implémenter la fonction `doFilterInternal`, en récupérant le token bearer dans la requête, récupérer l'utilisateur grâce au token, re-créer un `UsernamePasswordAuthenticationToken` et le connecter dans le `SecurityContextHolder`.
Attention: Dans le `UsernamePasswordAuthenticationToken` vous devez stocker le rôle de l'utilisateur en tant que `GrantedAuthority`, il faut donc ajouter une méthode `getAuthorities` dans votre modèle `JwtUserDetails` qui retourne le rôle de l'utilisateur au format attendu.

f) Ajout d'un point d'entrée

Dans le même package, ajoutez une nouvelle classe `JwtAuthenticationEntryPoint` qui implémente `AuthenticationEntryPoint` et `Serializable`. La méthode `commence` doit envoyer une erreur disant que l'utilisateur n'est pas autorisé.

g) Configuration de l'application

Dernière étape, vous devez créer la classe `ApplicationConfig` dans le package `config`.

Vous devez ajouter une première fonction `userDetailsService()` qui utilise votre classe `jwtUserDetailsService` pour retourner un `UserDetailsService` via le username de l'utilisateur.

Afin de sécuriser les mots de passe, nous allons ajouter un chiffrement via **bcrypt** dans un Bean `passwordEncoder()`. Une fois que c'est configuré, il faut modifier la fonction d'enregistrement (register) pour chiffrer le mot de passe !

Vous devez également définir un Bean `authenticationProvider` qui retourne un `DaoAuthenticationProvider` dans lequel vous devez y définir votre `userDetailsService` et votre `passwordEncoder`.

La dernière fonction à implémenter est `authenticationManager` qui retourne un `AuthenticationManager` chargé via un getter depuis la configuration passée en paramètre de cette fonction.

h) Configuration de la sécurité

Dernière étape, vous devez créer la classe `SecurityConfig` dans le package `config`. Cette dernière ne doit pas étendre d'une autre classe mais doit avoir les annotations nécessaires pour activer la sécurité http sur les contrôleurs.

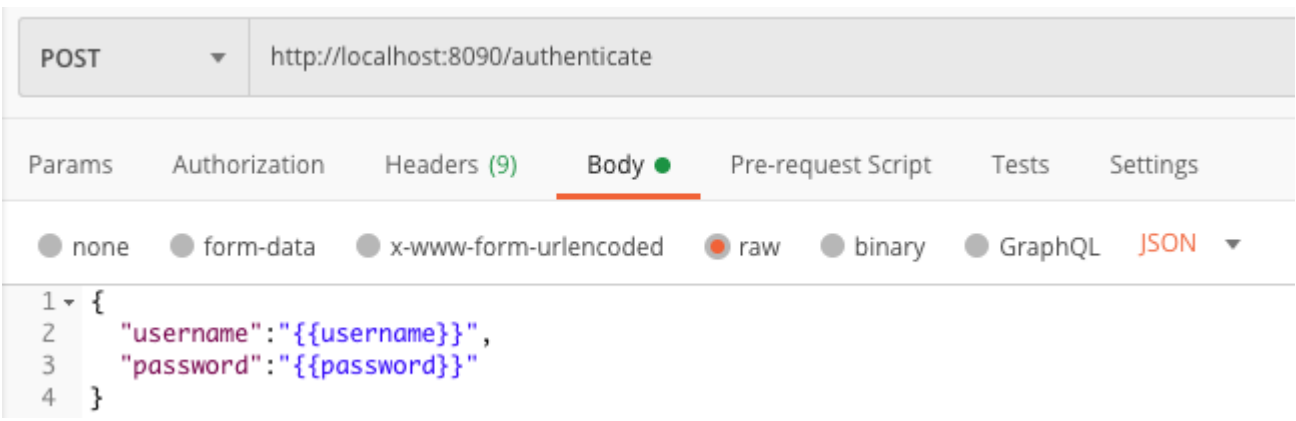
Vous devez y configurer la fonction `securityFilterChain(HttpSecurity http)` qui contiendra toute la sécurité de vos routes. Vous devez y désactiver la vérification du CSRF et autoriser les accès aux routes `register` et `authenticate` sans restrictions.

Vous devez également y ajouter l'entrypoint que nous venons de créer grâce à `exceptionHandling` et l'authentication provider via `authenticationProvider`.

En dernier, ajoutez le filtre `jwtRequestFilter` qui doit se lancer avant d'exécuter le contrôleur via `addFilterBefore`.

3) Ajout d'une fonction d'authentification utilisateur

Si tout est bien configuré, vous pouvez récupérer votre Bearer Token en postant vos données sur l'URL suivante : http://localhost:8090/authenticate

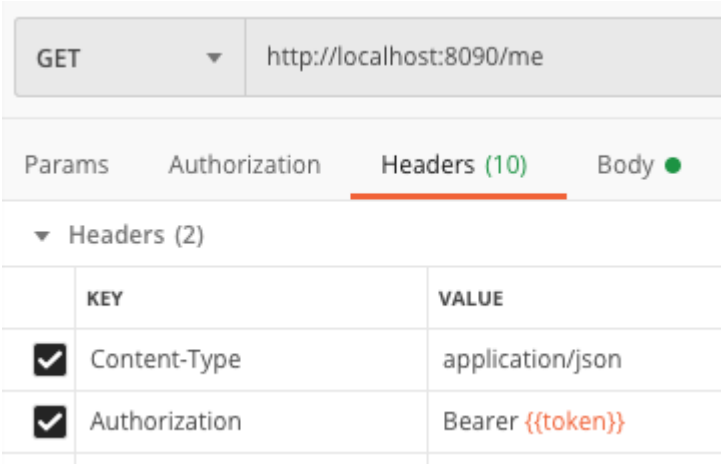


Voici les cas de réponse à respecter :

Cas	Code HTTP	Réponse (toujours JSON)
Succès	200	Retourne un JSON avec une clé `token` uniquement
Mauvais utilisateur / mot de passe	401	JSON avec un message d'erreur

4) Ajout d'une fonction `/me`

Vous allez maintenant ajouter au contrôleur `AuthenticationController` une route `/me` répondant aux requêtes `GET`. Cette route doit être protégée par l'authentification que vous venez de mettre en place.



Voici les cas de réponse à respecter :

Cas	Code HTTP	Réponse (toujours JSON)
Succès	200	Retourne un objet `UserDetails`
Requête invalide	400	JSON avec un message d'erreur
JWT invalide ou absent	401	JSON avec un message d'erreur

En cas de succès, la fonction renvoie un objet `UserDetails` correspondant à l'utilisateur actuellement connecté. Vous ne devez pas retourner le mot de passe de l'utilisateur, même chiffré.