Spring Boot Rest + Spring Security

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Spring Boot Rest & Spring Security

Objectif

- Sécuriser l'accès à nos ressources Rest

Ce qu'il faut

- Spring Security

Spring Boot Rest & Spring Security

Création de projet Spring Boot

- Aller dans `File > New > Other`
- Chercher `Spring`, dans `Spring Boot` sélectionner `Spring Starter Project` et cliquer sur `Next >`
- Saisir
  - `SpringBootRestSecurity` dans `Name`,
  - `com.example` dans `Group`,
  - `SpringBootRestSecurity` dans `Artifact`
  - `com.example.demo` dans `Package`
- Cliquer sur `Next >`
- Chercher et cocher les cases correspondantes aux `Spring Data JPA`, `MySQL Driver`, `Spring Web` et `Spring Security` puis cliquer sur `Next >`
- Valider en cliquant sur `Finish`

Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `puclic static void main`) est `com.example.demo`
- Tous les autres packages `dao`, `model`... doivent être dans le package `demo`.

Et si on part d'un projet existant, il faudra ajouter les dépendances suivantes

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Étapes

- Créer une classe `SecurityConfig` pour la configuration de la sécurité dans un package `com.example.demo.security`
- Copier les packages `com.example.demo.dao` et `com.example.demo.model` du projet associé au chapitre précédent dans le `src/main/java` de ce projet
- Copier aussi le contenu d'`application.properties` du projet associé au chapitre précédent dans le `src/main/resources` de ce projet
- Créer ou copier les entités, `User` et `Role` et les deux classes, `UserDetailsImpl` et `UserDetailsServiceImpl`, du projet associé au chapitre **Spring Security** dans le `com.example.demo.security` de ce projet

Spring Boot Rest & Spring Security

Dans `com.example.demo.security`**, définir la classe** `SecurityConfig`

```
package com.example.demo.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.authentication.
   builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.
   HttpSecurity;
import org.springframework.security.config.annotation.web.configuration
   .EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration
   .WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService
   ;
import org.springframework.security.crypto.password.NoOpPasswordEncoder
   ;
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{
```

Spring Boot Rest & Spring Security

Suite de `SecurityConfig`

```
@Autowired
private UserDetailsService userDetailsService;
@Bean
public static NoOpPasswordEncoder passwordEncoder() {
  return (NoOpPasswordEncoder) NoOpPasswordEncoder.getInstance();
}

@Autowired
public void configure(AuthenticationManagerBuilder auth) throws
   Exception {
  auth.userDetailsService(userDetailsService).passwordEncoder(
     passwordEncoder());
}

@Override
protected void configure(HttpSecurity http) throws Exception {
  http.cors().and()
     .authorizeRequests().anyRequest().fullyAuthenticated();
  http.httpBasic();
  http.csrf().disable();
}
```

Contenu de l'entité `Role`

```
@Entity
@Table(name="roles")
public class Role {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY
     )
  private Long id;
  private String titre;

  // ajouter les getters / setters
```

Contenu de l'entité `User`

```
@Entity
@Table(name = "users")
public class User implements Serializable {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long num;
   private String userName;
   private String password;
   @ManyToMany(cascade={CascadeType.PERSIST, CascadeType.
      REMOVE},fetch = FetchType.EAGER)
   private List <Role> roles = new ArrayList<Role>();
   // + getters & setters + addRole & removeRole
```

`fetch = FetchType.EAGER` pour indiquer que la relation doit être chargée en même temps que l'entité `User`.

Contenu de `UserRepository`

```
public interface UserRepository extends JpaRepository<User,
   Long> {
  public User findByUserName(String username);
}
```

La méthode `findByUserName` sera utilisée plus tard.

Contenu de `RoleRepository`

```
public interface RoleRepository extends JpaRepository<Role,
   Long>{

}
```

Supprimons `@RepositoryRestResource` **dans** `PersonneRepository`

```
public interface PersonneRepository extends JpaRepository<
   Personne, Long>{
}
```

Ajouter l'annotation aux contrôleurs REST

```
@CrossOrigin
@RestController
public class PersonneRestController {

  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/personnes")
  public List<Personne> getPersonnes() {
    return personneRepository.findAll();
  }

  @GetMapping("/personnes/{id}")
  public Personne getPersonne(@PathVariable("id") long id) {
    return personneRepository.findById(id).orElse(null);
  }

  @PostMapping("/personnes")
  public Personne addPersonne(@RequestBody Personne personne) {
    return personneRepository.save(personne);
  }
}
```

Créer une classe qui implémente l'interface `UserDetailsService`

```
package com.example.demo.security;

import org.springframework.stereotype.Service;

@Service
public class UserDetailsServiceImpl implements
   UserDetailsService {
}
```

Cette classe implémente l'interface `UserDetailsService`, elle doit donc implémenter la méthode `loadUserByUserName()` qui retourne un objet de la claase qui implémente l'interface `UserDetails`.

On utilise l'annotation Service pour dire qu'il s'agit d'un `Component` qu'on peut injecter avec l'annotation `Autowired` et qui fait parti de la couché métier.

Spring Boot Rest & Spring Security

Créer une classe qui implémente l'interface `UserDetailsService`

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetailsService
   ;
import org.springframework.security.core.userdetails.
   UsernameNotFoundException;
import org.springframework.stereotype.Service;
import com.example.demo.dao.UserRepository;
import com.example.demo.model.User;
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
  @Autowired UserRepository userRepository;
  @Override
  public UserDetailsImpl loadUserByUsername(String username) throws
     UsernameNotFoundException {
    User user=userRepository.findByUserName(username);
    if(null == user){
      throw new UsernameNotFoundException("No user named "+username);
    }else{
      return new UserDetailsImpl(user);
    }
  }
}
```

Créer une classe qui implémente l'interface `UserDetails`

```
package com.example.demo.security;

public class UserDetailsImpl  implements UserDetails
   {
}
```

Cette classe implémente l'interface `UserDetails`, elle doit donc implémenter toutes les méthodes que nous détaillerons dans la slide suivante

Spring Boot Rest & Spring Security

Créer une classe qui implémente l'interface `UserDetails`

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.
   SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import com.example.demo.model.Role;
import com.example.demo.model.User;
public class UserDetailsImpl  implements UserDetails{
  private User user;
  public  UserDetailsImpl(User user){
    this.user = user;
  }
  @Override
  public Collection<? extends GrantedAuthority> getAuthorities() {
    final List<GrantedAuthority> authorities = new ArrayList<
       GrantedAuthority>();
    for (final Role role : user.getRoles())
      authorities.add(new SimpleGrantedAuthority(role.getTitre()));
    return authorities;
  }
```

```
  @Override
  public boolean isAccountNonExpired() {
    return true;
  }
  @Override
  public boolean isAccountNonLocked() {
    return true;
  }
  @Override
  public boolean isCredentialsNonExpired() {
    return true;
  }
  @Override
  public boolean isEnabled() {
    return true;
  }
  @Override
  public String getUsername() {
    return user.getUserName();
  }
  @Override
  public String getPassword() {
    return user.getPassword();
  }
}
```

Dans `application.properties`, on ajoute les données concernant la connexion à la base de données et la configuration de `Hibernate`

```
spring.datasource.url = jdbc:mysql://localhost:3306/myBase?useUnicode=
  true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&
  serverTimezone=UTC
spring.datasource.username =root
spring.datasource.password =root
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql = true
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.
  MySQL5Dialect
```

Cette partie (`?useUnicode=true&useJDBCCompliantTimezoneShift=true &useLegacyDatetimeCode=false&serverTimezone=UTC`) dans la chaîne de connexion est ajoutée pour éviter un bug du connecteur `MySQL` concernant l'heure système.

L'ajout de la propriété `spring.jpa.hibernate.naming.physical-strategy = org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl` permet de forcer **Hibernate** à utiliser les mêmes noms pour les tables et les colonnes que les entités et les attributs.

Avant de tester, créer trois utilisateurs avec des rôles différents

```
insert into roles values (1,"ROLE_ADMIN"),
(2,"ROLE_USER");

insert into users values (1,"wick","wick"),
(2,"john","john"),
(3,"alan","alan");

insert into users_roles values (1,1),
(2,2),
(1,2),
(3,1);
```

Pour tester

Il faut aller sur `http://localhost:8080/personnes`, s'authentifier avec les identifiants d'un utilisateur de la table `users`

Pour ajouter une personne

- utiliser Postman en précisant la méthode et l'url `http://localhost:8080/personnes`
  - dans `Headers`, préciser la clé `Content-Type` et la valeur `application/json`
  - dans `Body`, cocher `raw` et sélectionner `JSON(application/json)`

Exemple de valeurs à persister

```
{
    "nom": "dalton",
    "prenom": "jack"
}
```

Exercice 1

Tester, avec Postman, les trois méthodes **HTTP** `put` et `delete` qui permettront de modifier ou supprimer une personne.

Exercice 2

Créer une application **Angular** qui permet à un utilisateur, via des interfaces graphiques) la gestion de personnes (ajout, modification, suppression, consultation et recherche) en utilisant les web services définis par **Spring**.

Spring Boot Rest & Spring Security

Pour Angular, il faut activer le CORS en définissant cette nouvelle classe de configuration

```
package com.example.demo.security;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.
   WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

  @Override
  public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
      .allowedMethods("HEAD", "GET", "PUT", "POST", "DELETE", "PATCH",
        "OPTIONS")
      .allowedHeaders("*")
      .exposedHeaders("WWW-Authenticate")
      .allowCredentials(true)
      .allowedOrigins("*")
      .maxAge(TimeUnit.DAYS.toSeconds(1));
  }
}
```

Dans le service `personne.service.ts`, il faut ajouter les identifiants de l'utilisateur à l'entête de la requête HTTP

```
add(personne){
    let username: string = 'wick';
    let password: string = 'wick';
    let user = username+":"+password;
    const headers = new HttpHeaders().set('Authorization', "
       Basic␣" + btoa(user));
    console.log("Basic␣" + btoa(user))
    return this.http.post(this.url, personne, { headers:
       headers });
}
```

Les imports nécessaires

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
```