

# Spring security

Spring Sécurité est un framework permettant d'ajouter aux applications Spring authentification et contrôle d'accès.

Spring security ajoute au Dispatcher servlet de Spring MVC un ensemble de filtres (servlet filters) qualifié de **filter chain**.

## Intégration

Ajouter les dépendances suivantes à **pom.xml** :

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

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
</dependency>
```

## Authentification par défaut

L'intégration de Spring security met en place une authentification basique par défaut, avec un utilisateur de non **user**, ayant le rôle **USER**, dont le password s'affiche dans la console de démarrage de Spring, utilisable en mode développement :

```
Using generated security password: 90990698-6bb0-4953-8151-fe8010f2547a
```

```
This generated password is for development use only. Your security configuration must be updated before running your application in production.
```



Il est possible de modifier l'utilisateur par défaut dans **application.properties** :

```
spring.security.user.name=boris  
spring.security.user.password=boris-password  
spring.security.user.roles=manager
```

## Configuration

La configuration se fait au travers d'une classe de configuration activant la sécurité :

```
@Configuration  
@EnableWebSecurity  
class WebSecurityConfig {  
    @Bean  
    @Throws(Exception::class)  
    fun configure(http: HttpSecurity): SecurityFilterChain {  
        http  
            .csrf().disable().authorizeHttpRequests()  
            .requestMatchers("/master/**").hasRole("manager")  
            .anyRequest().authenticated()  
            .and()  
            .formLogin()  
        return http.build()  
    }  
}
```

## Authentification depuis une BDD

### Entity

Créer une entity **User** pour gérer les utilisateur et leur rôle :

```
@Entity
open class User() {

    constructor(username:String):this(){
        this.username=username
    }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    open var id:Int=0

    @Column(length = 65, nullable = false)
    open lateinit var username:String

    @Column(length = 256)
    open var password:String?=null

    @Column(unique = true, length = 115)
    open var email:String?=null

    @Column(nullable = false)
    open lateinit var role:String
}
```

Créer le repository associé et ajouter une méthode permettant de rechercher un User par son username ou son email :

```
interface UserRepository : JpaRepository<User, Int> {
    @Query("SELECT u FROM User u WHERE u.username=:usernameOrEmail OR u.email=:usernameOrEmail")
    fun findByUsernameOrEmail(usernameOrEmail: String?): User?
}
```

## UserDetailsService

Créer un Service implémentant **UserDetailsService** :

Ce service retourne depuis la base de données un utilisateur recherché par son nom ou son email et retourne une instance de **UserDetails** :

```
class DbUserService:UserDetailsService {
    @Autowired
    lateinit var userRepository: UserRepository

    @Autowired
    lateinit var passwordEncoder: PasswordEncoder

    override fun loadUserByUsername(username: String?): UserDetails {
        val user= userRepository.findByUsernameOrEmail(username!!) ?: throw UsernameNotFoundException("User not found")
        return
    }
}
```

```
org.springframework.security.core.userdetails.User(user.username,user.password,
getGrantedAuthorities(user))
    }

    private fun getGrantedAuthorities(user: User): List<GrantedAuthority>? {
        val authorities: MutableList<GrantedAuthority> = ArrayList()
        authorities.add(SimpleGrantedAuthority(user.role))
        return authorities
    }

    fun encodePassword(user: User) {
        user.password=passwordEncoder.encode(user.password)
    }
}
```

## Password encoder

Modifier la configuration de spring security :

```
@Configuration
@EnableWebSecurity
class WebSecurityConfig {

    @Bean
    @Throws(Exception::class)
    fun configure(http: HttpSecurity): SecurityFilterChain {
        ...
    }

    @Bean
    fun userDetailsService(): UserDetailsService? {
        return DbUserService()
    }

    @Bean
    fun bCryptPasswordEncoder(): BCryptPasswordEncoder? {
        return BCryptPasswordEncoder()
    }
}
```

From:  
<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:  
<http://slamwiki2.kobject.net/web/framework/spring/security?rev=1678562629>

Last update: **2023/03/11 20:23**

