Spring framework 5 - Spring security

Acceder a la infromación del rol

Spring permite acceder a la información tanto del usuario como del rol a través de una clase llamada **SecurityContextHolder**. A continuación se muestra un ejemplo:

```
Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
  Collection<? extends GrantedAuthority> roles =
authentication
    .getAuthorities();
  log.info("Roles {}",roles.toString());
```

Los roles serán modificados por Spring agregando el prefijo ROLE a continuación un ejemplo:

ROLE_ADMIN

Method security

En los ejemplos anteriores controlamos el acceso a los *controllers* de nuestra aplicación, pero al realizar seguridad podemos ir más allá y restringir el acceso a los métodos de acuerdo a un rol.

Configuración

Para configurar la seguridad a nivel de método agregaremos la siguiente clase a nuestra aplicación:

```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true,
                            securedEnabled = true,
                            jsr250Enabled = true)
public class GlobalSecirutyConfig extends
WebSecurityConfigurerAdapter  {
}
```

Del código anterior podemos observar lo siguiente:

-**prePostEnabled** : Habilita las anotaciones **@PreAuthorize** y **@PostAuthorize** entre otras.
-**securedEnabled**: Habilita la anotación **@Secured**
-**jsr250Enabled** : Habilita la anotación **@RoleAllowed**

Las anotaciones anteriores nos permitirán realizar autorización en nuestra aplicación a nivel de método.

@Secured

Anotación de Spring que permite definir la lista de roles permitidos al ejecutar un método, por ejemplo:

```
@Secured({"ROLE_ADMIN","ROLE_USER"})
public List<String> getAll() {
      log.info("Getting all ");
      return personajes;
}
```

La definición anterior indica que solo los usuarios cuyas credenciales contengan el rol ADMIN o USER podrán ejecutar el método getAll().

@RolesAllowed

La anotación **@RolesAllowed** es equivalente a utilizar **@Secured,** la diferencia es que la primera no es propia de Spring, se recomienda utilizarla en caso de que se considere la migración de Spring a otra tecnología, ejemplo:

```
@RolesAllowed({"ROLE_ADMIN","ROLE_USER"})
public List<String> getAll() {
      log.info("Getting all ");
      return personajes;
}
```

La definición anterior indica que solo los usuarios cuyas credenciales contengan el rol ADMIN o USER podrán ejecutar el método **getAll**().

@PreAuthorize / @PostAuthorize

Las anotaciones **@PreAuthorize** y **@PostAuthorize** permiten utilizar ***expression-based access control*** a través de SpEL.

La anotación **@PreAuthorize** permite realizar una evaluación antes de ejecutar el método, dependiendo del resultado de la evaluación decide si **ejecutarlo** o no.

La anotación **@PostAuthorize** permite realizar una evaluación después de ejecutar el método, dependiendo del resultado de la evaluación decide **devolver** su valor o no.

A continuación se muestra un ejemplo:

```
 @PreAuthorize("hasRole('ROLE_USER') or
hasRole('ROLE_ADMIN')")
 @PostAuthorize("hasRole('ROLE_ADMIN')")
 public List<String> getAll() {
       log.info("Getting all ");
       return personajes;
 }
```

La definición anterior indica que tanto el rol **USER** como el rol **ADMIN** pueden ejecutar el método **getAll**(), pero solo el rol **ADMIN** puede obtener su respuesta.

@PostAuthorize es utilizado para el caso en el que los valores a utilizar se tomen en tiempo de ejecución y se desee validar si el objeto puede ser accedido por el usuario.

Meta anotaciones

Es posible crear una meta anotación para combinar el uso de las anotaciones de seguridad. A continuación el ejemplo anterior utilizando meta anotaciones:

```
@Retention(RUNTIME)
@PreAuthorize("hasRole('ROLE_USER') or
hasRole('ROLE_ADMIN')")
@PostAuthorize("hasRole('ROLE_ADMIN') and 7 <8")
public @interface IsUserOrAdmin {
}
```

La anotación **@IsUserOrAdmin** tendrá el mismo efecto que aplicar las anotaciones **@PreAuthorize** y **@PostAuthorize** mostradas.

A continuación un ejemplo sobre su uso:

```
@IsUserOrAdmin
public List<String> getAll() {
      log.info("Getting all ");
      return personajes;
}
```

Status HTTP

En caso de que se cubra la autenticación pero no la autorización, se genera un error como el siguiente:

```
{
   "timestamp": "2019-06-28T19:20:01.162+0000",
   "status": 403,
   "error": "Forbidden",
   "message": "Forbidden",
}
```

Este error indica que el usuario tuvo una atenticación exitosa pero que falló al realizar la autorización.

Anotación a nivel de clase

Si consideramos aplicar alguna anotación de seguridad a todos los métodos, es preferible definirla a nivel de clase:

```
@Service
@IsUserOrAdmin
public class PersonajesService {
}
```

DEVS4J
Hecho por Devs para Devs

Testing de aplicaciones seguras

Una vez entendido cómo construir aplicaciones seguras, el siguiente paso será probar aplicaciones seguras. Para esto debemos asegurarnos de tener en nuestro proyecto la siguiente dependencia:

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

Test de integración

Al tener nuestra dependencia de *testing* definida analicemos el siguiente test:

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class PersonajesServiceTest {

      @Autowired
      private PersonajesService personajesService;

       @Test(expected=AccessDeniedException.class)
      @WithMockUser(username="raidentrance",
      password="devs4j",authorities="OWNER")
      public void testServiceWithWrongCredentials(){
            List<String> personajes =
      personajesService.getAll();
      }
      @Test
      @WithMockUser(username="raidentrance",
      password="devs4j",authorities="ROLE_ADMIN")
      public void testServiceWithGoodCredentials(){
            List<String> personajes =
      personajesService.getAll();
            assertThat(personajes).size().isEqualTo(30);
      }
}
```

Del *test* anterior podemos analizar los siguientes puntos:

-Utilizamos la anotación **@SpringBootTest** para iniciar un contexto de Spring de *testing* y autoconfigurar nuestra aplicación.

-Utilizamos la anotación **@WithMockUser**() para simular el usuario autenticado al ejecutar el método.

-En el primer *test* se puede ver que el usuario tiene el rol de **OWNER** y que el método **getAll**() solo permite usuarios con rol **ROLE_ADMIN** y **ROLE_USER**, por esto se espera una excepción de tipo AccessDeniedException.

-En el segundo *test* se puede ver que el usuario tiene el rol de **ROLE_ADMIN** y este es uno de los roles permitidos en el método **getAll**() por esto se busca validar que la respuesta tenga un total de 30 registros, los cuales son poblados en el método anotado con **@PostConstruct**.

-Si se desea que todos los *tests* utilicen el mismo usuario de prueba, la anotación **@WithMockUser** se debe colocar a nivel de clase.

-*Spring security test* permite definir un **UserDetails** personalizado, probar con meta anotaciones, entre muchas otras funcionalidades.