Seção 03 - Resumo

Consulta por usuário

Quando um usuário cadastrado tentar logar na aplicação, será necessário testar suas credenciais. O Spring Security precisa apenas que criemos um método de consulta pelo login do usuário, sem que precisemos testar a senha. A senha será testada pelo próprio Spring Security, a partir do objeto `Usuario`, retornado pela consulta.
Sendo assim, a consulta é muito básica e como usamos o e-mail cadastrado como login (username), basta usarmos esse filtro para localizar o usuário em questão no banco de dados:

```
@Query("select u from Usuario u where u.email like :email")
Usuario findByEmail(@Param("email") String email);
```

Se a consulta retornar um resultado significa que o usuário existe, então, vamos pegar o retorno da consulta e passar para o Spring Security e ele vai então se encarregar de testar se a senha digitada pelo usuário equivale a senha armazenada no banco de dados.
Para realizar esse processo vamos precisar implementar o método `loadUserByUsername()`, pertencente a interface `UserDetailsService` do Spring Security. Tal interface deve ser implementada na classe `UsuarioService`, conforme o exemplo a seguir:

```
@Service
public class UsuarioService implements UserDetailsService {
  @Autowired
  private UsuarioRepository repository;

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      Usuario usuario = repository.findByEmail(username);
      return new User(
               usuario.getEmail(),
               usuario.getSenha(),
               AuthorityUtils
                   .createAuthorityList(getAuthorities(usuario.getPerfis()))
           );
       }
}
```

Observe que o método `loadUserByUsername()` recebe como argumento um objeto String que representa o nome de usuário digitado pelo usuário no formulário de login. Este é o valor que vamos passar como parâmetro na consulta `findByEmail()`.
O retorno da consulta será um objeto `Usuario` e devemos usar este objeto em um segunda etapa dentro do método `loadUserByUsername()`.

Nessa segunda etapa devemos preparar o retorno do método `loadUserByUsername()`, que por definição do Spring Security, deve ser um objeto do tipo `UserDetails`.
Como `UserDetails` é uma interface, vamos retorno uma classe que a implementa, nesse caso a classe `User`.

Em uma instancia de `User` vamos informar três parâmetros que são o nome de usuário, a senha e a lista de perfis. Esses parâmetros estarão presentes no objeto `Usuario` retornado pela consulta.

Como trabalhamos com uma lista de perfis do tipo `java.util.List<Perfil>` e o método construtor de `User` espera por uma lista do tipo `Collection<? extends GrantedAuthority>`, será necessário usar o método `createAuthorityList()` da classe `AuthorityUtils`.

Este método espera por um array do tipo `java.lang.String` e será necessário realizar uma pequena adaptação para atribuir corretamento o parâmetro esperado pelo método. Para isso, vamos criar um método que converta o objeto `java.util.List<Perfil>` em um `java.lang.String[]`:

```
private String[] getAuthorities(List<Perfil> perfis) {
    String[] authorities = new String[perfis.size()];
    for (int i = 0; i < perfis.size(); i++) {
       authorities[i] = perfis.get(i).getDesc();
    }
   return authorities;
}
```

Esse método de conversão será o `getAuthorities()` e a conversão é bastante simples como é possível observar nas instruções apresentadas. Basicamente, percorremos a lista de perfis e recuperamos a descrição de cada perfil. Essas descrições são então adicionas em um array de strings que será retornado pelo método.

Lidando com a senha

Após a consulta pelo usuário o Spring Security vai ter acesso ao objeto `UserDetais` retornado pelo método `loadUserByUsername()`. A partir dai, o Spring Security vai ter acesso a senha retornada pelo banco de dados e ele poderá testar se essa senha confere com a senha digitada pelo usuário no formulário de login.
Entretanto, a senha do banco de dados estará criptografada e a senha digitada pelo usuário não. Portanto, será preciso que o Spring Security use o mesmo sistema de criptografia, utilizado para criptografar a senha no evento de cadastro de um novo usuário. Só assim, vai ser possível que o Spring Security compare as duas senhas.

Mas como o Spring Security vai saber qual o tipo de criptografia que foi usado para criptografar a senha durante o cadastro do usuário? Bem, devemos dizer isso a ele. E vamos dizer isso a partir de um método na classe de configuração.

O método em questão será sobrescrito da classe `WebSecurityConfigurerAdapter` e se chama `configure()`, porém, desta vez usaremos o método com o argumento do tipo `AuthenticationManagerBuilder`, como visto no código a seguir:

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(service)
        .passwordEncoder(new BCryptPasswordEncoder());
}
```

Observe que a variável `auth` da acesso ao método `userDetailsService`, o qual espera como parâmetro um objeto do tipo `UserDetailsService`. Por conta disso, injetamos na classe uma dependência de `UsuarioService` para passar o parâmetro desejado, já que `UsuarioService` é um `UserDetailsService`.

Na sequencia, teremos acesso ao método `passwordEncoder()`, o qual será usado para dizer ao Spring Security qual a criptografia que deve ser processada. Nesse projeto usaremos a BCrypt, atualmente a mais recomendada. Para esse tipo de criptografia o Spring Security fornece a classe `BCryptPasswordEncoder`. Dessa forma, o Spring Security saberá que precisa comparar as senhas por meio da criptografia BCrypt.

Atualmente, além da BCrypt o Spring Security fornece classes de criptografias do tipo SCrypt (`SCryptPasswordEncoder`) e Pbkdf2 (`Pbkdf2PasswordEncoder`).

Referencias

- Spring Security - https://spring.io/projects/spring-security#learn

Código Fonte

Caso tenha tido algum tipo de dificuldade para acompanhar a desenvolvimento do código fonte até o final desta seção, ele está disponível na área de arquivo para download.