Curso: Spring Boot com Angular 7

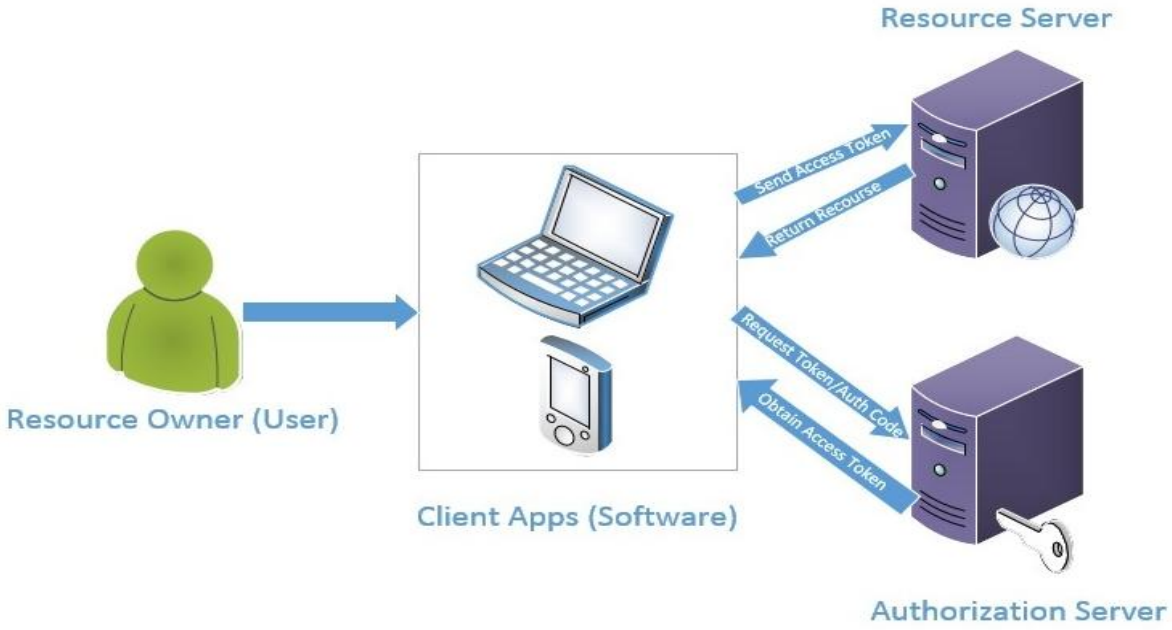
https://www.udemy.com/user/hugo-silva

Prof. Hugo Silva

Capítulo: Autenticação e Autorização com Spring Security Oauth2

Objetivo geral:

- Criar e configurar um serviço de autenticação de autorização e usuário
- Criar Refresh Token de usuário
- Testar a API REST usando Access Token



Configuração Inicial do Spring Security

Checklist:

- Em **pom.xml**, incluir as dependências do **Spring Security:**

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

- Criar o pacote **security** e a classe **WebSecurityConfiguration**
  - Esta classe deve herdar de **WebSecurityConfigurerAdapter**
- Para adaptar esta classe ao **Spring Security Oauth2**, precisa criar os seguintes métodos:

```
@Bean
public BCryptPasswordEncoder bCryptPasswordEncoder() {}
```

```
@Bean
@Override
public AuthenticationManager authenticationManagerBean() throws Exception {}
```

```
@Bean
DaoAuthenticationProvider authenticationProvider() {}
```

```
@Override
protected void configure(AuthenticationManagerBuilder auth){}
```

Adaptando o nosso serviço de autenticação

Checklist:

- No pacote **service** iremos criar a classe ***CustomUserDetailsService*** *implements* ***UserDetailsService***
- Atualizar a Classe **User** com:
  - Criar os atributos **password** e **enabled**,
  - Criar o construtor ***User(User user)***
  - Modificar o construtor User(...)

```
public User(String id,String firstName, String lastName, String email, String
password, boolean enabled) { ...}
```

- Atualizar o método **Update** da Classe **UserService**;
- Atualizar a classe de carga inicial (classe SetupDataLoader) da base de dados com os novos atributos
- Implementar a classe ***GrantedAuthority*** na classe **Role**
- Ainda na classe ***CustomUserDetailsService*** deve criar a classe:

```
private final static class UserRepositoryUserDetails extends User implements
UserDetails {}
```

- Implementar estes métodos na classe ***CustomUserDetailsService***

```
@Override
public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {}
```

```
private final List<GrantedAuthority> getGrantedAuthorities(final
Collection<Role> roles) {}
```

```
public final Collection<? extends GrantedAuthority> getAuthorities(final
Collection<Role> roles){}
```

- Atualizar o método ***authenticationProvider()*** *da classe* ***WebSecurityConfiguration*** com classe ***CustomUserDetailsService***

```
@Bean
public DaoAuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
    provider.setPasswordEncoder( bCryptPasswordEncoder() );
    provider.setUserDetailsService(userDetailsService);
```

```
    return provider;
}
```

Criando a Authorization Service da Oauth2

Checklist:

- Em pom.xml, incluir as dependências do Spring Security oauth2:

```
<dependency>
   <groupId>org.springframework.security.oauth</groupId>
   <artifactId>spring-security-oauth2</artifactId>
   <version>2.3.4.RELEASE</version>
</dependency>
```

- No pacote security.oauth2 criar a classe ***AuthorizationServerConfiguration*** *extends* ***AuthorizationServerConfigurerAdapter***

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws
Exception{}
```

```
@Override
public void configure(ClientDetailsServiceConfigurer clients ) throws
Exception{}
```

```
@Bean
@Primary
public DefaultTokenServices tokenServices() {}
```

Criando a Resource Service da Oauth2

Checklist:

- No pacote **security.oauth2** criar a classe ***public class ResourceServerConfiguration*** *extends* ***ResourceServerConfigurerAdapter***
- Nesta classe deve criar os seguintes métodos:

```
@Override
public void configure(ResourceServerSecurityConfigurer resources) {}
```

```
@Override
public void configure(HttpSecurity http) throws Exception {}
```

Configurando os CORSFilter

Checklist:

- No pacote config criar a classe ***public class CORSFilter*** *implements* ***Filter***
- Implementar apenas o métodos:

```
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Component;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class CORSFilter implements Filter {

@Override
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
throws IOException, ServletException {
    HttpServletResponse response = (HttpServletResponse) res;
    HttpServletRequest request = (HttpServletRequest) req;
    response.setHeader("Access-Control-Allow-Origin", "*");
    response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS,
DELETE, PUT");
    response.setHeader("Access-Control-Max-Age", "3600");
    response.setHeader("Access-Control-Allow-Headers", "x-requested-with,
authorization, x-auth-token, origin, content-type, accept");

    if ("OPTIONS".equalsIgnoreCase(request.getMethod())) {
        response.setStatus(HttpServletResponse.SC_OK);
    } else {
        chain.doFilter(req, res);
    }
}
}
```

Nota: **A Filter que deve ser implementada do tipo *import javax.servlet.*;***

Referências:

http://software.dzhuvinov.com/cors-filter-spec.html
https://www.w3.org/TR/cors/

Testando os Endpoint's usando Access Token

Checklist:

- Testar autenticação de usuário e **Security Oauth2** através do **Postman**
- Autenticação de usuário:
  - http://localhost:8080/oauth/token?grant_type=password&username=joao@gmail.com&password=123
- Testar os endpoint's do tipo GET:
  - http://localhost:8080/api/users e Bearer + Acces_Token
  - http://localhost:8080/api/users/{ID} e Bearer + Acces_Token
  - http://localhost:8080/api/users/{ID}/roles e Bearer + Acces_Token

- Testar os endpoint's do tipo POST:
  - http://localhost:8080/api/users e Bearer + Acces_Token
    {
      "firstName": "Carlos",

```
  "lastName": "Souza",
  "email": "carlos@gmail.com",
 "password": "123"
}
```

- Testar o endpoint do tipo PUT:
  - http://localhost:8080/api/users/{ ID } e Bearer + Acces_Token

```
{
  "firstName": "Mario",
  "lastName": "Santana",
  "email": "mario@gmail.com"
}
```

- Testar endpoint DELETE:
  - http://localhost:8080/api/users/{ID} e Bearer + Acces_Token

Refresh Token

Checklist:

- Mudar o tempo de access token para um minuto e refresh token para 24 horas **configure(ClientDetailsServiceConfigurer clients)**

```
.accessTokenValiditySeconds(60)
.refreshTokenValiditySeconds(60*60*24);
```

- Testar Refresh Token do **Security Oauth2** através do **Postman**
  - http://localhost:8080/oauth/token?grant_type=password&username=joao@gmail.com&password=123
  - http://localhost:8080/oauth/token?grant_type=refresh_token&refresh_token={REFRESH_TOKEN}