1 OAuth Spring Boot

Es wurde eine Spring-Boot Anwendung implementiert, die *OAuth* Authentifizierungen über *Facebook* und über einen eigens implementierten *Authorization Server* in *.Net (Identity Foundation)*. Es werden auch als *Open Source Authorization Server* angeboten wie z.B. *Keycloak* von der *Apache Software Foundation*, der in Java und AngularJS implementiert ist.

Die *Spring Security* kann vollständig transparent von der eigentlichen Anwendung konfiguriert werden. Die geschützten Ressourcen können gebündelt in einer Konfigurationsklasse definiert werden, ohne dass die Implementierungen der eigentlichen Anwendung geändert müssen. Als Benutzerrepräsentation sollte eine eigene Klasse implementiert werden die *Security*-spezifische Implementierung von der Anwendung abstrahiert, damit kann die gewählte Authentifizierungsmethode in Hintergrund ausgetauscht werden.

Listing 1: Spring application.properties

```
# Logging setup
#logging.level.root=DEBUG
logging.level.org.springframework.security=DEBUG

# thymeleaf setup
spring.thymeleaf.cache=false
spring.thymeleaf.check-template-location=true
spring.thymeleaf.prefix=classpath:templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.mode=HTML5
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.content-type=text/html

facebook.client.clientId=1749583525361910
facebook.client.clientSecret=6195e3b92e6032a33fafcb75a1682945
facebook.client.accessTokenUri=https://graph.facebook.com/oauth/access_token
facebook.client.userAuthorizationUri=https://www.facebook.com/dialog/oauth
facebook.client.tokenName=oauth_token
facebook.client.authenticationScheme=query
facebook.client.clientAuthenticationScheme=form
facebook.resource.userInfoUri=https://graph.facebook.com/me

identityserver.client.clientId=sfsclient
identityserver.client.clientSecret=sfsclient
identityserver.client.accessTokenUri=http://localhost:5000/connect/token
identityserver.client.userAuthorizationUri=http://localhost:5000/connect/authorize
identityserver.client.tokenName=oauth_token
identityserver.client.scope=openid profile role
identityserver.client.authenticationScheme=query
identityserver.client.clientAuthenticationScheme=form
identityserver.resource.userInfoUri=http://localhost:5000/connect/userinfo
```

In der Datei *application.properties* werden die verwendeten *OAuth Authentication Server* konfiguriert, jedoch nicht die geschützten Ressourcen, diese werden in einer Spring Konfigurationsklasse definiert.

Diese Methode ist Teil der programmatischen Konfiguration der *Spring Security* der Klasse *SecurityConfiguration* und konfiguriert die *HttpSecurity* Instanz.

```
/**
* Configure the htt security for the spring application
*
* @param http the HttpSecurity object to configure
* @throws Exception if an error occurs during the configuration
*/
@Override
protected void configure(HttpSecurity http) throws Exception {
        // Enable OAuth for all resources
        http.antMatcher("/**").authorizeRequests()
        // Exclude open resources such as css, js and open views
        .antMatchers("/", "/dist/**", "/vendor/**", "/login**", "/landing")
                .permitAll()
        // protect all admin views for role 'ADMIn' only
        .antMatchers("/admin/**").hasRole("ADMIN").anyRequest().authenticated()
        // Redirect to login page if error
        .and().exceptionHandling()
                .authenticationEntryPoint(new LoginUrlAuthenticationEntryPoint("/"))
        // Redirect to login page if successfully logged out
        .and().logout().logoutSuccessUrl("/").permitAll()
        // Enable cross site request forgery support
        .and().csrf().csrfTokenRepository(CookieCsrfTokenRepository
                .withHttpOnlyFalse())
        // Add the OAuth filter before the BasicAuthentication filter
        .and().addFilterBefore(ssoFilter(), BasicAuthenticationFilter.class);
}
```

Mit dieser Methode wird der *OAuth-Servlet-Filter* registriert und wird als erster Filter aktiviert *(ordinal=-100)*.

```
    /**
* Register OAuth filter
*
* @param filter the filter to append
* @return the filter registration
*/
@Bean
public FilterRegistrationBean
        oauth2ClientFilterRegistration(OAuth2ClientContextFilter filter) {
        FilterRegistrationBean registration = new FilterRegistrationBean();
        registration.setFilter(filter);
        registration.setOrder(-100);
        return registration;
}
```

Die folgende Methode konfiguriert einen *CompositeFilter*, der mehrere Filter Instanzen verwaltet und als ein Filter registriert wird. In unserer Implementierung werden die OAuth-Authentifizierung über *Facebook* und unseren *Identity Server* unterstützt.

```
   /**
* Helper method for configuring the BasicAuthentication filter instance.
*
* @return the configured filter instance
*/
private Filter ssoFilter() {
        CompositeFilter filter = new CompositeFilter();
        List<Filter> filters = new ArrayList<>(2);

        // Configure the OAuth filter for facebook authentication
        OAuth2ClientAuthenticationProcessingFilter facebookFilter =
                new OAuth2ClientAuthenticationProcessingFilter("/login/facebook");
        OAuth2RestTemplate facebookTemplate =
                new OAuth2RestTemplate(facebook(), oauth2ClientContext);
        facebookFilter.setRestTemplate(facebookTemplate);
        facebookFilter.setTokenServices(
                new UserInfoTokenServices(facebookResource().getUserInfoUri(),
                                          facebook().getClientId()));
        facebookFilter.setAuthenticationSuccessHandler(
                new SimpleUrlAuthenticationSuccessHandler("/home"));
        filters.add(facebookFilter);

        // Configure the OAuth filter for identity server authentication
        OAuth2ClientAuthenticationProcessingFilter identityServerFilter =
                new OAuth2ClientAuthenticationProcessingFilter("/login/identityserver");
        OAuth2RestTemplate identityServerTemplate =
                new OAuth2RestTemplate(identityserver(), oauth2ClientContext);
        identityServerFilter.setRestTemplate(identityServerTemplate);
        identityServerFilter.setTokenServices(
                new UserInfoTokenServices(identityserverResource().getUserInfoUri(),
                                          identityserver().getClientId()));
        identityServerFilter.setAuthenticationSuccessHandler(
                new SimpleUrlAuthenticationSuccessHandler("/home"));
        filters.add(identityServerFilter);

        // Set configured filters on the composite filter
        filter.setFilters(filters);

        return filter;
}
```

Die folgenden Methoden erstellen die Java-Repräsentation der in der *application.properties* definierten Properties *facebook.** und *identityserver.**. In diese Objekte werden die gesetzten *Properties* in der*application.properties* sowie die programmatischen Konfigurationen zusammengeführt.

```
    /**
* @return the configured facebook client
*/
@Bean
@ConfigurationProperties("facebook.client")
public AuthorizationCodeResourceDetails facebook() {
        return new AuthorizationCodeResourceDetails();
}

/**
* @return the configured facebook resource
*/
@Bean
@ConfigurationProperties("facebook.resource")
public ResourceServerProperties facebookResource() {
        return new ResourceServerProperties();
}

/**
* @return the configured identity server client
*/
@Bean
@ConfigurationProperties("identityserver.client")
public AuthorizationCodeResourceDetails identityserver() {
        return new AuthorizationCodeResourceDetails();
}

/**
* @return the configured identity server resource
*/
@Bean
@ConfigurationProperties("identityserver.resource")
public ResourceServerProperties identityserverResource() {
        return new ResourceServerProperties();
}
```

Mit dieser Konfiguration ist die *Security* der implementierten *Spring Boot* Anwendung konfiguriert und die Ressourcen der Anwendung sind gemäß der Konfiguration geschützt.