Spring Breakout Session
Spring Security

by Vlad Costel Ungureanu
for FII UAIC, Learn Stuff

Spring Breakout Session

Spring Overview

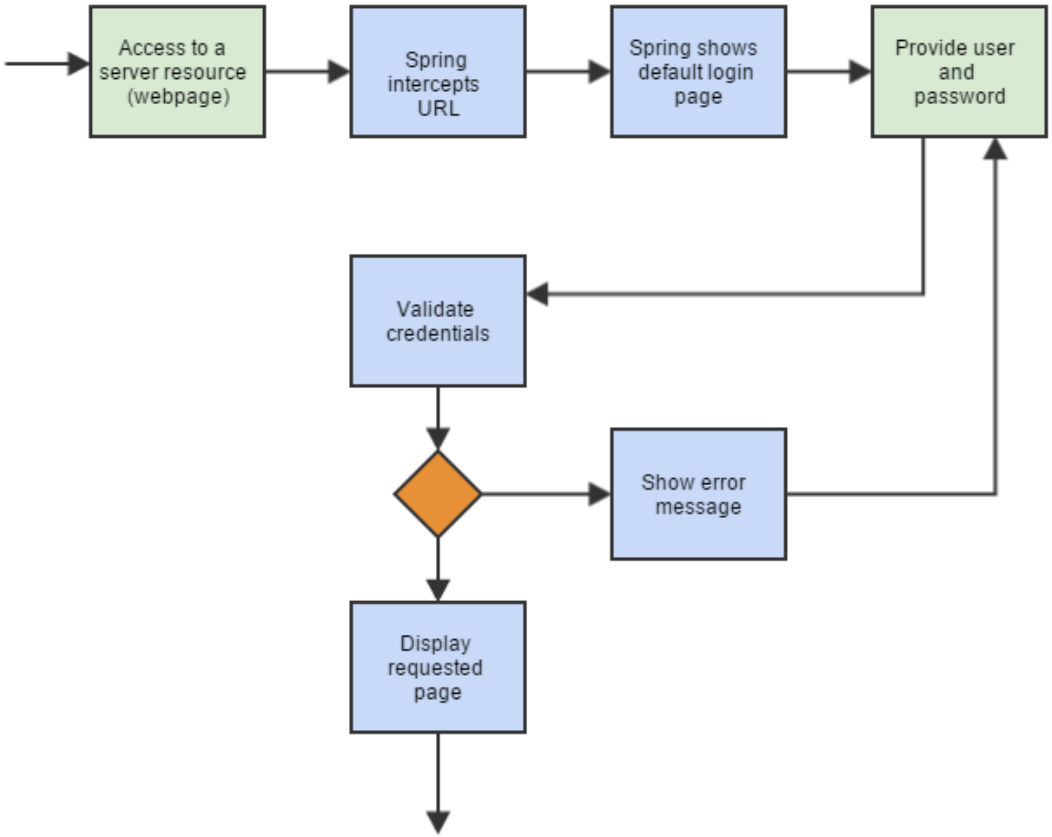
- Spring Security
- Configuration
- Standard Approach
- Encrypt Password
- User Service Provider
- CSRF Form
- CSRF AJax
- Security Chain Filters
- Spring OAuth2
- Overview

LEARN stuff IO

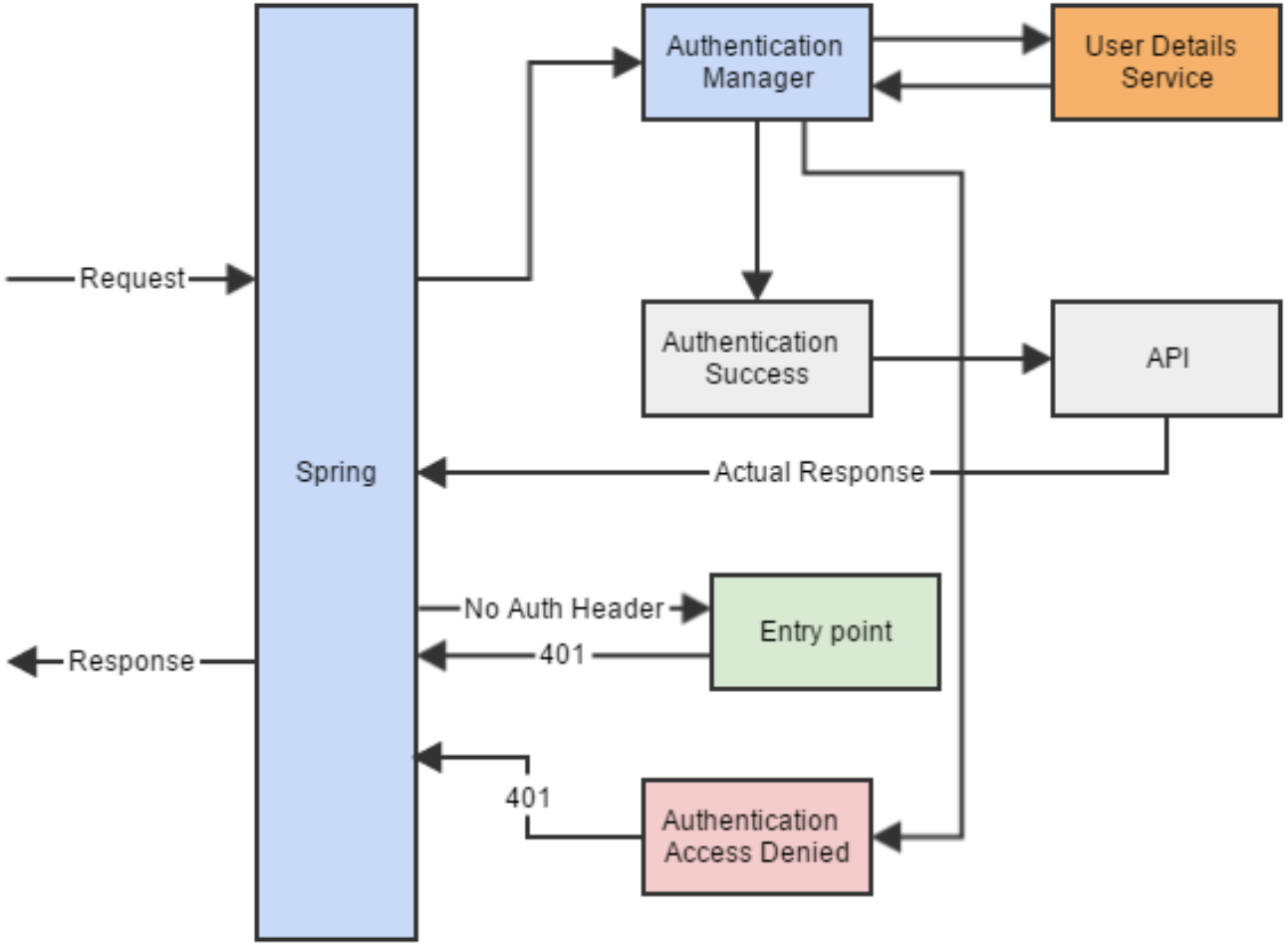
Spring Security

- Almost all applications contain data that is sensitive or important for the user, even though it may not be secret
- While security is mandatory for sensitive or secret data it is our responsibility to make sure that user data is secured, that information is not altered or lost and that
- The application should only permit a specific user access specific data
- Application administrators should have other privileges compared to normal users
- Paying user should have access to paid features
- Data exchange should always happen between user and application without a third party listening in

Spring Security – Web Apps



Spring Security – Rest Services



Configuration

```
<security:http auto-config="true" use-expressions="true">
    <security:intercept-url pattern="/login“ access="permitAll" />
    <security:intercept-url pattern="/resources/**“ access="permitAll" />
    <security:intercept-url pattern="/login/**“ access="permitAll" />
    <security:intercept-url pattern="/**"
      access="hasAnyRole('ROLE_USER','ROLE_ADMIN', 'ROLE_GUEST')" />
    <security:form-login login-page="/login"
      authentication-failure-url="/login?error" default-target-url="/home"
      always-use-default-target="true" />
    <security:logout logout-url="/j_spring_security_logout"
      logout-success-url="/login" />
    <security:logout delete-cookies="JSESSIONID" />
    <!-- enable csrf protection -->
    <security:csrf />
  </security:http>
```

Standard

```
<beans:bean id="passwordEncoder"
        class="com.app.quiz.utils.BCryptPasswordEncoder" />
<security:authentication-manager>
        <security:authentication-provider>
        <security:password-encoder ref="passwordEncoder" />
        <security:jdbc-user-service
                                data-source-ref="dataSource"
                                users-by-username-query=
                                "select user_email as username, user_password as password,
        enabled as active from app_users where user_email=?"
                                authorities-by-username-query=
                                “select us.user_email as username, ug.group_name from
                app_users us,app_groups ug, app_user_groups ur where
        ug.id = ur.id_group and us.id = ur.id_user and us.user_email =? " />
        </security:authentication-provider>
</security:authentication-manager>
```

Encrypt Password

```
public class BCryptPasswordEncoder implements PasswordEncoder {

	@Override
	public String encode(CharSequence arg0) {
		return MD5Encode.encode("" + arg0);
	}

	@Override
	public boolean matches(CharSequence arg0, String arg1) {
		return (MD5Encode.encode("" + arg0).equals(arg1)) ? true : false;
	}
}
```

Encrypt Password

```
public class MD5Encode {
        public static String encode(String password){
        String encoding = "";
        try{
                MessageDigest m = MessageDigest.getInstance("MD5");
                m.update((password.getBytes("UTF8")));
                byte s[] = m.digest();
                for (int i = 0; i < s.length; i++) {
                encoding += Integer.toHexString((0x000000ff & s[i]) | 0xffffff00).substring(6);
                }
        } catch(Exception e){
                e.printStackTrace();
        }
        return encoding;
} }
```

User Service Configuration

```
<beans:bean id="passwordEncoder"
      class="com.app.quiz.utils.BCryptPasswordEncoder" />

<security:authentication-manager>
      <security:authentication-provider user-service-ref="userServiceImpl">
         <security:password-encoder ref="passwordEncoder" />
      </security:authentication-provider>
</security:authentication-manager>
```

User Service

```
@Service("userServiceImpl")
public class UserServiceImpl implements UserService, UserDetailsService {
@Override
   public UserDetails loadUserByUsername(String username) throws
        UsernameNotFoundException {
      User userDetails = userDAO.getUserByEmail(username);
      List<SimpleGrantedAuthority> authorities = new ArrayList<>();
      Group group = userDetails.getGroup();
      SimpleGrantedAuthority authority = new SimpleGrantedAuthority(
                                                group.getGroupName());
      authorities.add(authority);
      return new org.springframework.security.core.userdetails.User(
            userDetails.getUserEmail(), userDetails.getUserPassword(),
            userDetails.isEnabled(), true, true, true, authorities);
   }
}
```

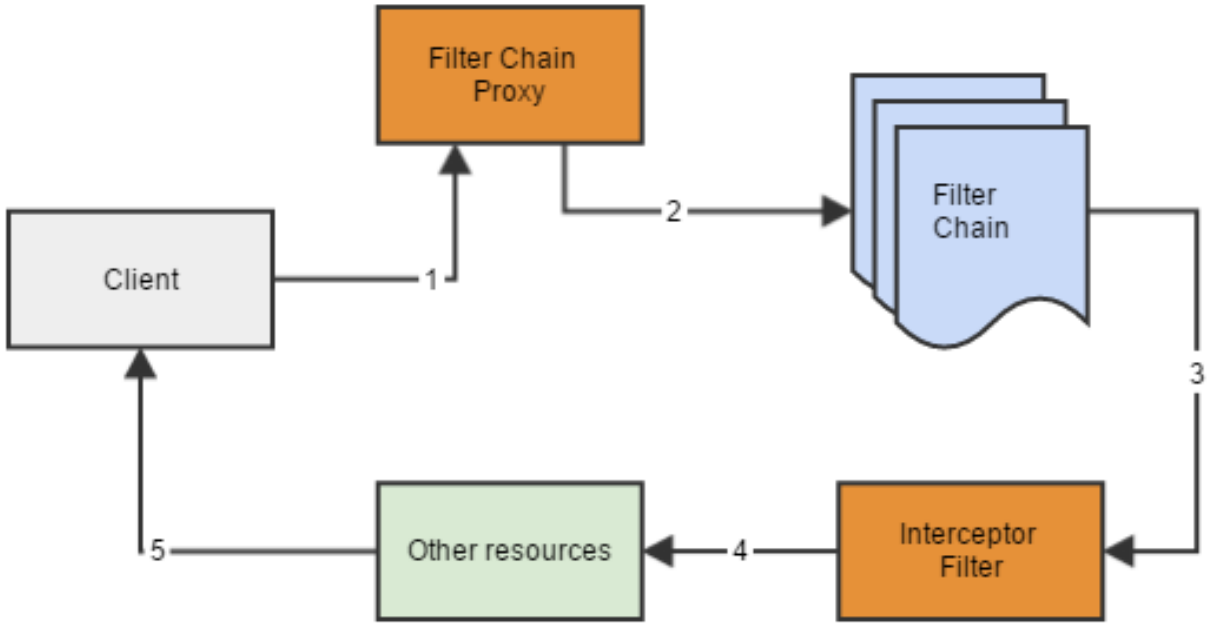
CSRF – Form submission

```
<head>
   <meta name="_csrf" content="${_csrf.token}"/>
   <meta name="_csrf_header" content="${_csrf.headerName}"/>
</head>

<c:url var="logoutUrl" value="/logout"/>
<form action="${logoutUrl}"
   method="post">
  <input type="submit"
   value="Log out" />
  <input type="hidden"
   name="${_csrf.parameterName}"
   value="${_csrf.token}"/>
</form>
```

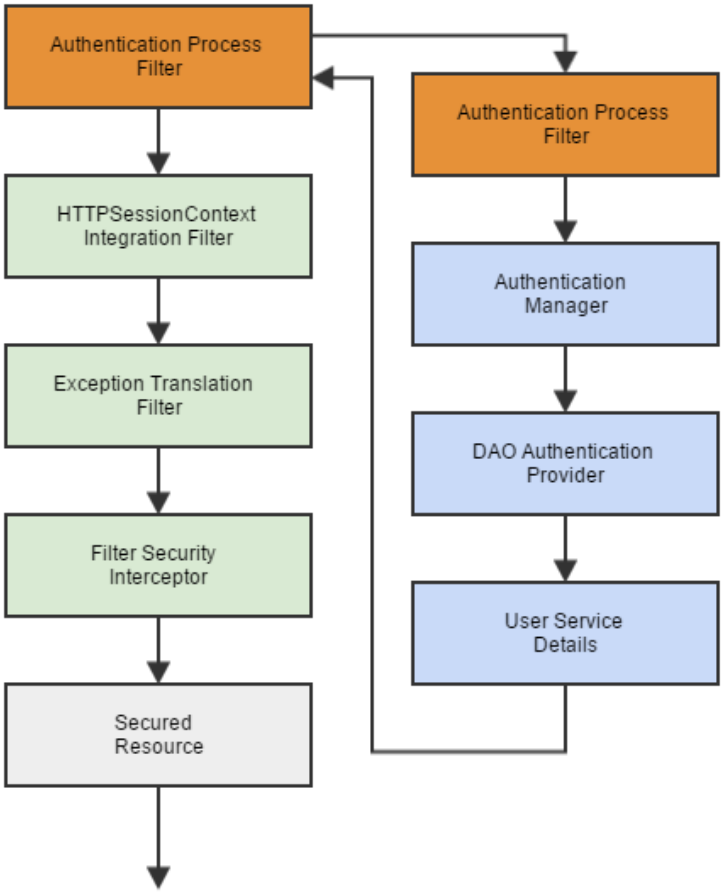
CSRF - Ajax

```
<head>
   <meta name="_csrf" content="${_csrf.token}"/>
   <meta name="_csrf_header" content="${_csrf.headerName}"/>
</head>

$(function () {
  var token = $("meta[name='_csrf']").attr("content");
  var header = $("meta[name='_csrf_header']").attr("content");
  $(document).ajaxSend(function(e, xhr, options) {
    xhr.setRequestHeader(header, token);
  });
});
```

Security Chain Filter



Security Chain Filter



Security Chain Filter

```
<bean id="filterChainProxy" class="org.springframework.security.web.FilterChainProxy">
  <sec:filter-chain-map path-type="ant">
    <sec:filter-chain pattern="/webServices/**" filters="
        securityContextPersistenceFilterWithASCFalse,
        basicAuthenticationFilter,
        exceptionTranslationFilter,
        filterSecurityInterceptor" />
    <sec:filter-chain pattern="/**" filters="
        securityContextPersistenceFilterWithASCTrue,
        formLoginFilter,
        exceptionTranslationFilter,
        filterSecurityInterceptor" />
  </sec:filter-chain-map>
</bean>
```

LEARN stuff IO

Security Chain Filter

1. ChannelProcessingFilter, because it might need to redirect to a different protocol

2. SecurityContextPersistenceFilter, so a SecurityContext can be set up in the SecurityContextHolder at the beginning of a web request, and any changes to the SecurityContext can be copied to the HttpSession when the web request ends (ready for use with the next web request)

3. ConcurrentSessionFilter, because it uses the SecurityContextHolder functionality but needs to update the SessionRegistry to reflect ongoing requests from the principal

4. Authentication processing mechanisms - UsernamePasswordAuthenticationFilter, CasAuthenticationFilter, BasicAuthenticationFilter etc - so that the SecurityContextHolder can be modified to contain a valid Authentication request token

5. The SecurityContextHolderAwareRequestFilter, if you are using it to install a Spring Security aware HttpServletRequestWrapper into your servlet container

Security Chain Filter

6. RememberMeAuthenticationFilter, so that if no earlier authentication processing mechanism updated the SecurityContextHolder, and the request presents a cookie that enables remember-me services to take place, a suitable remembered Authentication object will be put there

7. AnonymousAuthenticationFilter, so that if no earlier authentication processing mechanism updated the SecurityContextHolder, an anonymous Authentication object will be put there

8. ExceptionTranslationFilter, to catch any Spring Security exceptions so that either an HTTP error response can be returned or an appropriate AuthenticationEntryPoint can be launched

9. FilterSecurityInterceptor, to protect web URIs and raise exceptions when access is denied

```
CREATE TABLE tys_users (
id SERIAL UNIQUE NOT NULL,
 username VARCHAR(50) NOT NULL,
 email VARCHAR(50),
 password VARCHAR(500),
 fullname VARCHAR(100),
 accept_terms boolean DEFAULT false,
 activated boolean DEFAULT true,
 activationkey VARCHAR(50) DEFAULT NULL,
 resetpasswordkey VARCHAR(50) DEFAULT NULL,
 PRIMARY KEY(id)
);

CREATE TABLE tys_authority (
id SERIAL UNIQUE NOT NULL,
 name VARCHAR(50) NOT NULL,
 PRIMARY KEY(id)
);

CREATE TABLE tys_users_authority (
 user_id INTEGER NOT NULL  REFERENCES tys_users (id) ON DELETE CASCADE,
 authority_id INTEGER NOT NULL REFERENCES tys_authority (id)  ON DELETE CASCADE
);

CREATE TABLE oauth_access_token (
 token_id VARCHAR(256) DEFAULT NULL,
 token bytea,
 authentication_id VARCHAR(256) DEFAULT NULL,
 user_name VARCHAR(256) DEFAULT NULL,
 client_id VARCHAR(256) DEFAULT NULL,
 authentication bytea,
 refresh_token VARCHAR(256) DEFAULT NULL
);
CREATE TABLE oauth_refresh_token (
 token_id VARCHAR(256) DEFAULT NULL,
 token bytea,
 authentication bytea
);
```

Spring OAuth 2

```
// AuthorizationServerConfiguration
@Override
  public void configure(AuthorizationServerEndpointsConfigurer  endpoints) throws Exception {
  endpoints.tokenStore(tokenStore()).authenticationManager(this.authenticationManager)
        .userDetailsService(userDetailsService);
}

@Override
   public void configure(ClientDetailsServiceConfigurer  clients) throws Exception {
   clients.inMemory().withClient("client_name")
        .authorizedGrantTypes("password",  "refresh_token").authorities("USER")
        .scopes("read",  "write").resourceIds("resource_id")
        .secret("client_secret").accessTokenValiditySeconds(30000);
}
```

Spring OAuth 2

```
@Bean @Primary
   public DefaultTokenServices tokenServices() {
     DefaultTokenServices  tokenServices  = new DefaultTokenServices();
     tokenServices.setSupportRefreshToken(true);
     tokenServices.setTokenStore(tokenStore());
     tokenServices.setAccessTokenValiditySeconds(30000);
     return tokenServices;
   }
 }
// ResourceServerConfiguration
   @Override
   public void configure(HttpSecurity http) throws Exception {
http.authorizeRequests().antMatchers("/book/**").hasRole("ADMIN").antMatchers("/api")
        .authenticated().antMatchers("/signup/**").permitAll()
        .antMatchers("/public/**").permitAll()
        .antMatchers("/your_app/signup").permitAll();
}
```

Spring OAuth 2

```
@Override
@Transactional
public UserDetails loadUserByUsername(final String credentials) {
  logger.info("Authenticating " + credentials + ".");
   userFromDatabase = userRepository.findByEmail(lowercaseCredentials);

Collection<GrantedAuthority> grantedAuthorities = new ArrayList<>();
  for (Authority authority : userFromDatabase.getAuthorities()) {
    GrantedAuthority grantedAuthority = new SimpleGrantedAuthority(authority.getName());
    grantedAuthorities.add(grantedAuthority);
  }
  return new org.springframework.security.core.userdetails.User(userFromDatabase.getUsername(),
     userFromDatabase.getPassword(), grantedAuthorities);
}
```

HEADERS FOR AUTHENTICATION

Authorization: Basic some_encoded_string_based_on_client_and_secret=

Content-Type: application/x-www-form-urlencoded

HEADERS FOR CALLS

Content-Type: application/json,

Authorization: Bearer authentication_token

LEARN stuff IO

Assignments

✓Secure the previous application using Spring Security

✓The about page should not require login

✓Use CSRF for REST services

✓Implement OAuth2 instead of standard Spring Security

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com

This is a free course from LearnStuff.io
– not for commercial use –