**Spring Security**

@RestController  
public class HomeController {  
  
 @GetMapping("/")  
 public String myData() {  
 return "Hello World";  
 }  
}

import org.springframework.security.core.userdetails.User;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
  
@Configuration  
@EnableWebSecurity  
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Bean  
 @Override  
 protected UserDetailsService userDetailsService() {  
 List<UserDetails> userDetails = new ArrayList<>();  
 userDetails.add(User  
 .*withDefaultPasswordEncoder*()  
 .username("tanuj")  
 .password("tanuj")  
 .roles("USER")  
 .build());  
  
 return new InMemoryUserDetailsManager(userDetails);  
 }  
}

**Getting from the Database**

@Data  
@Entity  
public class User {  
  
 @Id  
 private int id;  
 private String username;  
 private String password;  
}

public interface MyRepository extends JpaRepository<User, Integer> {  
 User findByUsername(String username);  
}

@Configuration  
@EnableWebSecurity  
public class AppSecurityDBConfig extends WebSecurityConfigurerAdapter {  
 */\*\*  
 \* This needs to connect to the Dao layer  
 \*/* @Autowired  
 @Qualifier("myUserDetailsService")  
 private UserDetailsService userDetailsService;  
  
 @Bean  
 public AuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService);  
 provider.setPasswordEncoder(NoOpPasswordEncoder.*getInstance*());  
  
 return provider;  
 }  
}

@Service  
public class MyUserDetailsService implements UserDetailsService {  
  
 @Autowired  
 private MyRepository myRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = myRepository.findByUsername(username);  
 if (user == null) throw new UsernameNotFoundException("User does Not Exists");  
  
 return new MyUserPrincipal(user);  
 }  
}

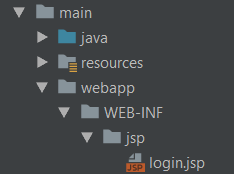
public class MyUserPrincipal implements UserDetails {  
  
 private User user;  
  
 public MyUserPrincipal(User user) {  
 this.user = user;  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return Collections.*singleton*(new SimpleGrantedAuthority("USER"));  
 }  
  
 @Override  
 public String getPassword() {  
 return user.getPassword();  
 }  
  
 @Override  
 public String getUsername() {  
 return user.getUsername();  
 }  
  
 @Override  
 public boolean isAccountNonExpired() { return true; }  
  
 @Override  
 public boolean isAccountNonLocked() { return true;}  
  
 @Override  
 public boolean isCredentialsNonExpired(){ return true; }  
  
 @Override  
 public boolean isEnabled(){ return true; }  
}

**FormLogin**

<dependency>  
 <groupId>org.apache.tomcat</groupId>  
 <artifactId>tomcat-jasper</artifactId>  
 <version>9.0.14</version>  
</dependency>

spring.mvc.view.prefix: /WEB-INF/jsp/  
spring.mvc.view.suffix: .jsp

@GetMapping("/login")  
public String login() {  
 return "login";  
}  
  
@GetMapping("/logout-success")  
public String logout() {  
 return "login";  
}

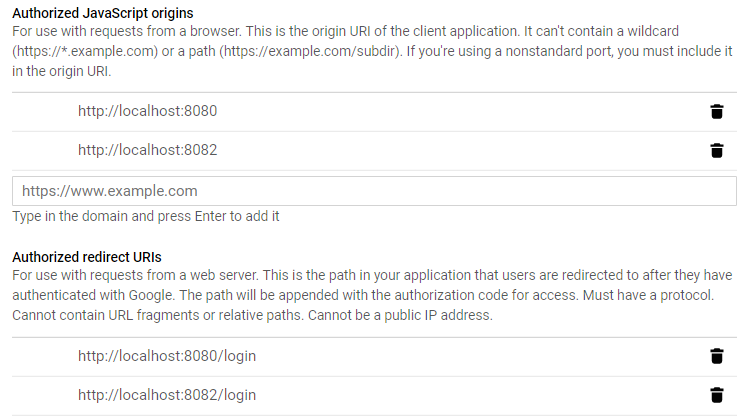


<body>  
<form action="login" method="post">  
 <h1>Login</h1>  
 ${SPRING\_SECURITY\_LAST\_EXCEPTION.message}  
 <table>  
 <tr>  
 <td>User:</td>  
 <td><input type="text" name="username"></td>  
 </tr>  
 <tr>  
 <td>Password:</td>  
 <td><input type="password" name="password"></td>  
 </tr>  
 <tr>  
 <td>Password:</td>  
 <td><input type="submit" name="submit" value="submit"></td>  
 </tr>  
 </table>  
</form>  
</body>  
</html>

@Configuration  
@EnableWebSecurity  
public class AppSecurityDBConfig extends WebSecurityConfigurerAdapter {  
  
 */\*\*  
 \* This needs to connect to the Dao layer  
 \*/* @Autowired  
 @Qualifier("myUserDetailsService")  
 private UserDetailsService userDetailsService;  
  
 @Bean  
 public AuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService);  
 provider.setPasswordEncoder(new BCryptPasswordEncoder());  
  
 return provider;  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf()  
 .disable()  
 .authorizeRequests()  
 .antMatchers("/login")  
 .permitAll()  
 .anyRequest()  
 .authenticated()  
 .and()  
 .formLogin()  
 .loginPage("/login")  
 .permitAll()  
 .and()  
 .logout().invalidateHttpSession(true)  
 .clearAuthentication(true)  
 .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))  
 .logoutSuccessUrl("/logout-success")  
 .permitAll();  
  
 }  
}

**Using Oauth Sso;**

In google you have to specify this in your App



So instead of having username and password from database you get these values from Authorization Server.

Benefits:

1. You don’t have to maintain the database to store username and password.
2. You can use Existing Authorization Servers like Facebook, Google, Github.
3. You take client id and client secret from these Authorization server.

***application.yml***

* **For *GOOGLE***

security:  
 oauth2:  
 client:  
 client-id: 972043024831-d8p0r35ohlurtvupse06hjrq3akjqi70.apps.googleusercontent.com  
 client-secret: alRCGD1ag0TIXADNBW\_lXz\_p  
 access-token-uri: https://www.googleapis.com/oauth2/v3/token  
 user-authorization-uri: https://accounts.google.com/o/oauth2/auth  
 token-name: oauth\_token  
 authentication-scheme: *query* client-authentication-scheme: *form* scope: profile email  
 resource:  
 user-info-uri: https://www.googleapis.com/userinfo/v2/me

* **For facebook**

security:  
 oauth2:  
 client:  
 clientId: 397420450803748  
 clientSecret: e0d437c8db20008d714e1128943e2966  
 accessTokenUri: https://graph.facebook.com/oauth/access\_token  
 userAuthorizationUri: https://www.facebook.com/dialog/oauth  
 tokenName: oauth\_token  
 authenticationScheme: *query* clientAuthenticationScheme: *form* resource:  
 userInfoUri: https://graph.facebook.com/me

***AppConfigOauth2Config***

@Configuration  
@EnableWebSecurity  
@EnableOAuth2Sso  
public class AppSecurityOauth2Config extends WebSecurityConfigurerAdapter {  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf()  
 .disable()  
 .authorizeRequests()  
 .antMatchers("/login")  
 .permitAll()  
 .anyRequest()  
 .authenticated();  
 }  
}

That’s it and you wont even have to configure Login/Logout functionality for you application. So now each time your client app starts it redirects you to google/facebook/github Authorization Server login page. Once you login to it with valida credentials, your app can start.

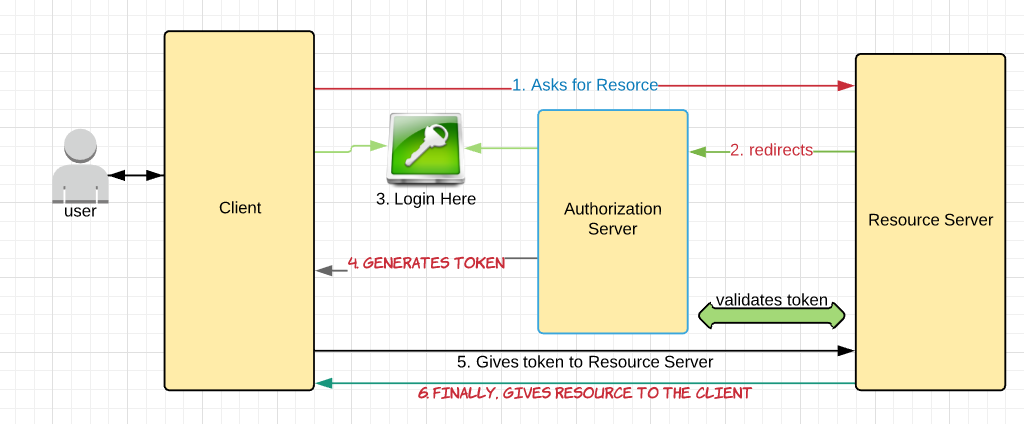
**Another way through Resource Server**

**security**:  
 **oauth2**:  
 **resource**:  
 **user-info-uri**: http://openreach\_dev-openreach\_dev.apigee.net/verifyoauth

@Configuration  
@EnableResourceServer  
@Profile("secured")  
class ResourceServerConfig {  
}

There are three roles in Oauth 2.0

1. User(Resource owner)
2. Application(Client)
3. API(Authorization server)



Suppose a website uses Facebook to signup or login.

Here, website is an application and Facebook is an API.

**Misconception:**

When you login to any website using Facebook or Google API that website grabs the username and password from Facebook or Google. That is wrong.

Actually, passwords are never passed from server to server in a Oauth 2.0 framework.

Suppose you have an Application named myApp. This application provides the bank balance of his/her account.

Any user is using this application on the internet clicks on the dashboard of myApp to know the account balance. So, myApp sends the authorization request to the Bank(authorization server). User will be prompted to authorize the request. Once user clicks on authorize the request authorization is granted to myApp and the Bank sends the authorization Id(Token) back to myApp.

Now that myApp has the Token it will send it to the Resource server so that it can access the Resource in which the information of the user is present. Resource server validates it against Authorization Server, and sends then sends the resource back to myApp.

In order to achieve above functionality, MyApp needs to register itself with the Bank(authorization server) API service.

MyApp needs to provide the Bank(authorization server) its name, website, and the URL to which the Bank(authorization server) will redirect once it authorizes the request.

**Step 1.**

When a Resource owner(User) wants to acces any confidential information on the client App(Web Application), client App in turn redirects the Resource owner with the endpoint of Authorization server.



Here, code is the one of the grant types.

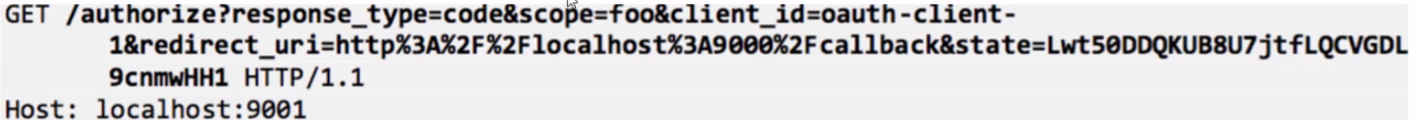
scope

client\_id: Given by authorization server when client was registerign to the A.S.

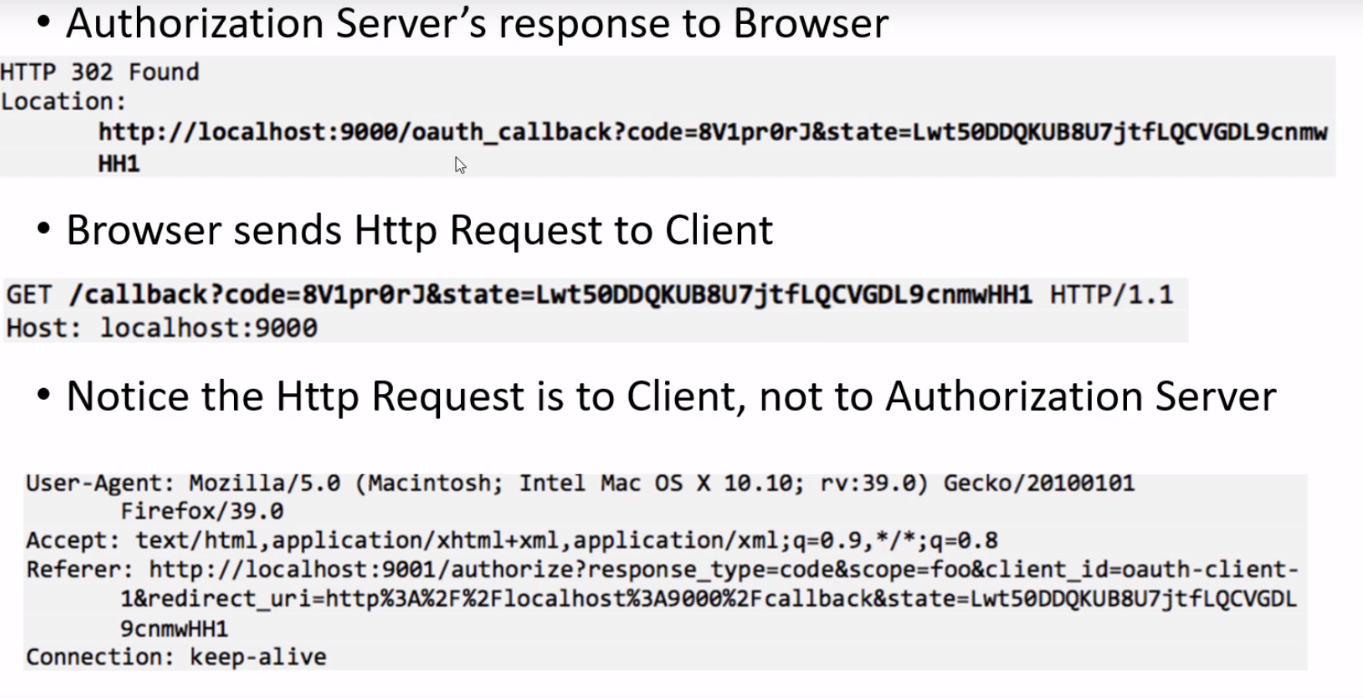
redirect\_uri

state: it is a unique sting and is a very important parameter, A.S will reject if it gets changed in between.

After this Resource owner sends GET request to A.s via client App.

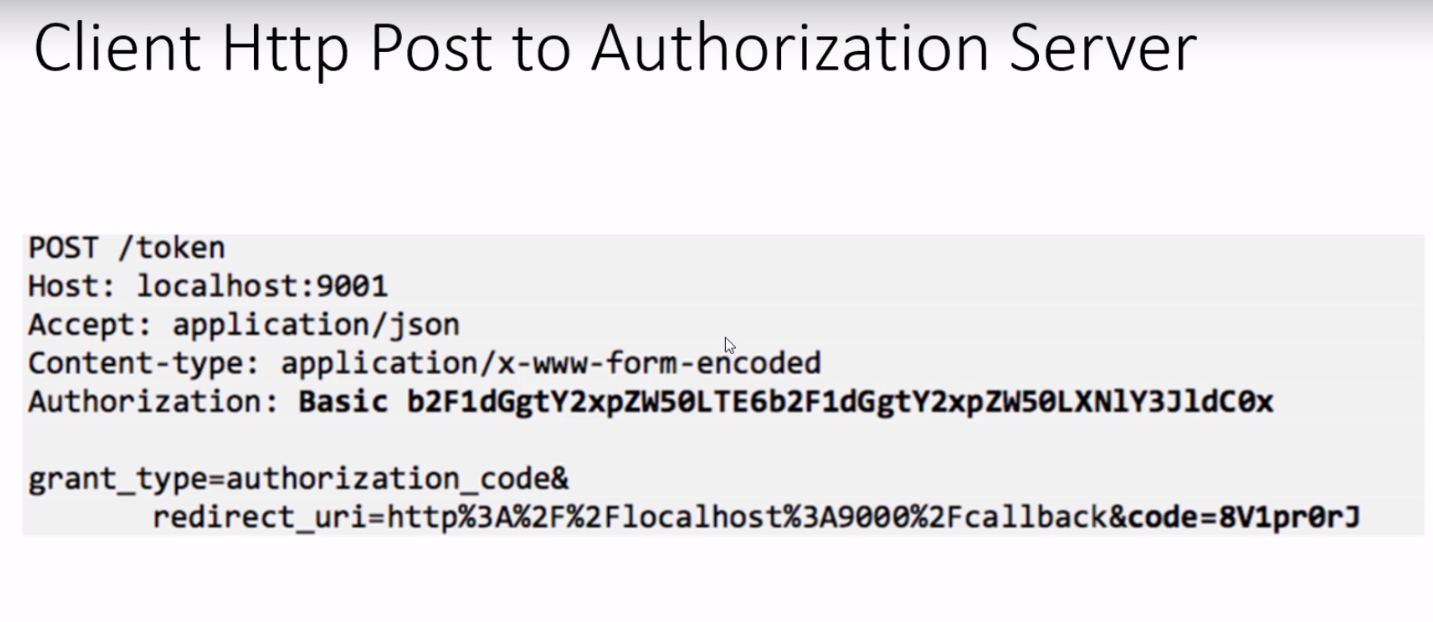


After this A.S responds back to resource owner via client app with authorization code and the same state as above:



Now, Client again sends this authorization code along with its credentials(Client\_Id and secret) to A.S again. Client sends in a POST form-encoded HTTP entity body.

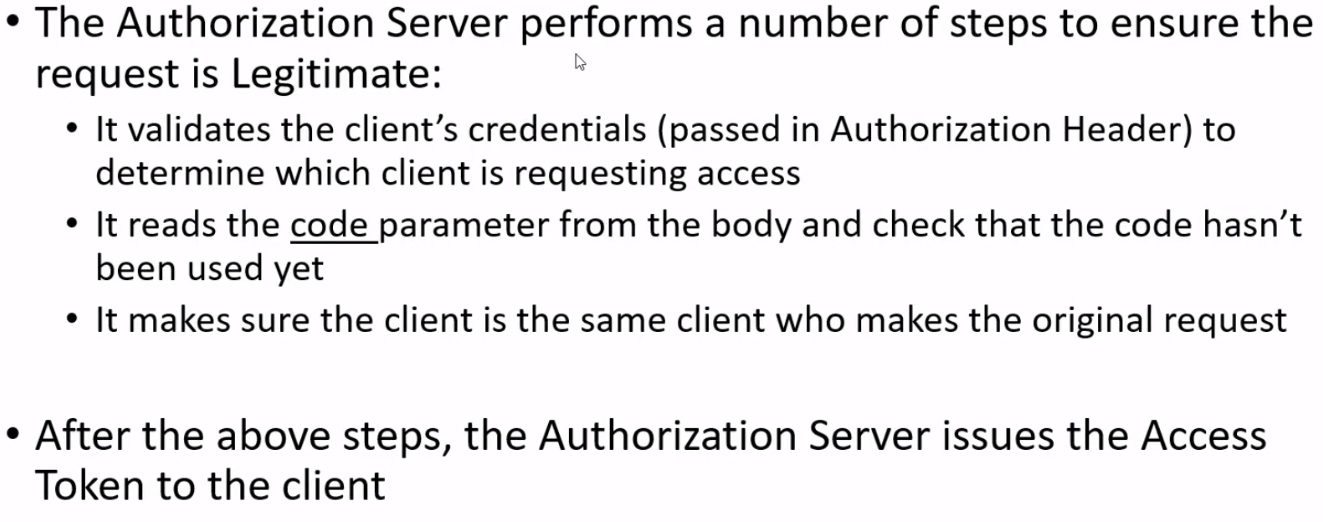
A.S decodes the header and knows which Client it is talking to.

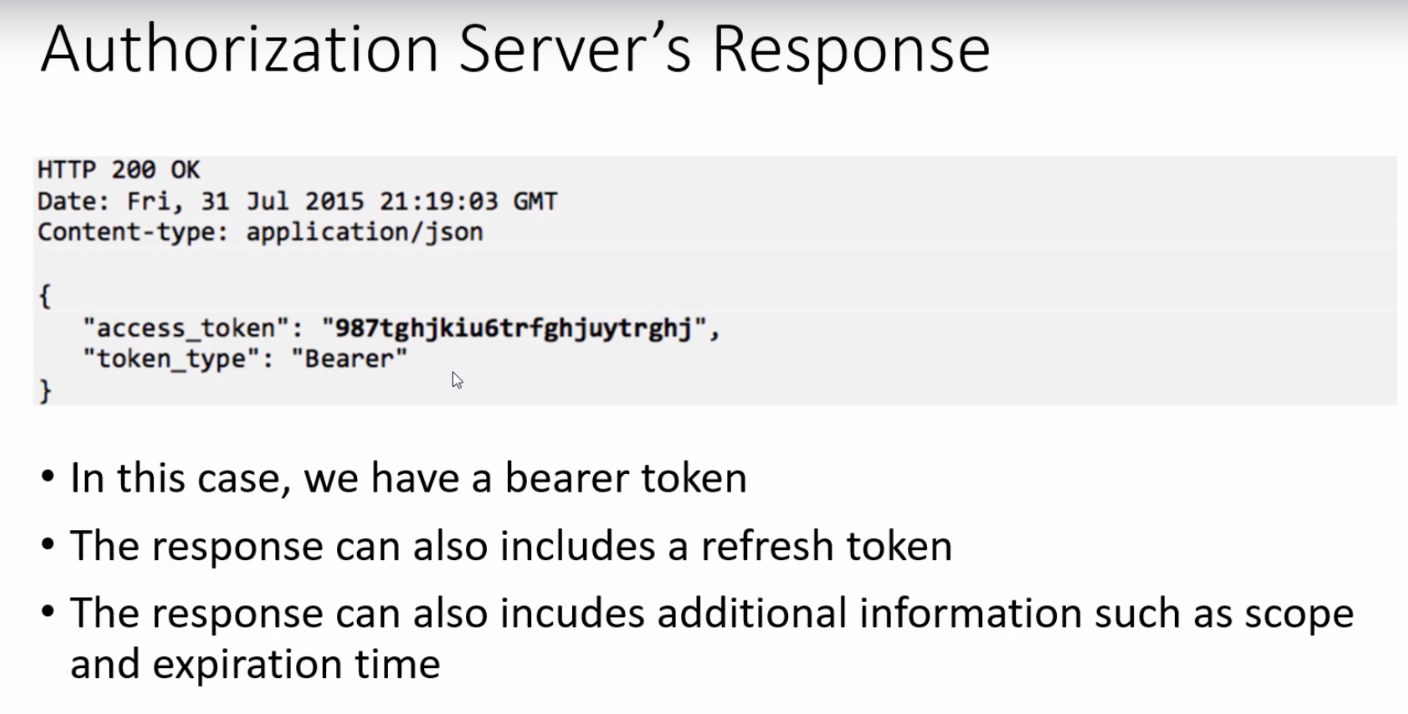


Above Authorization is a client\_Id with secret both encoded with 64 encoding.

Now, A.S issues Oauth Token to the client.

Issuing the Oauth token.





**Refresh Token:**

When client tries to access any resource from the resource server after getting the acess toke from the Authorization server and still gets an error in response then refresh token comes into the picture.

Client tries to refresh the access token from the authorization server which in turn returns the new access token and refresh token to the client.

Now, client again can request to the resource server with this refresh token.

**Authorization Grant:**

Authorization Grant component is when a user tries to grant a privilage to a client for which scope which it has to access the resource server.

In other words Oauth client is given an access to a protected resource using Oauth protocol by means of authorization grant.

Simply, Authorization grant is a method of getting a token.