Secure Spring REST API using OAuth2

Created on: July 30, 2016 | **Last updated on:** July 31, 2016 websystiqueadmin

Let's secure our `Spring REST API` using `OAuth2` this time, a simple guide showing what is required to secure a REST API using `Spring OAuth2`. Our use-case fits well with `Resource-owner Password Grant` flow of OAUth2 specification. We will use two different clients [Postman and a `Spring RestTemplate` based java application] to access our OAuth2 protected REST resources.
If you are already familiar with OAuth2 concepts, you may want to skip the theory, and jump right into code. As always, complete code can be found in attachment at the end of this article.

Other interesting posts you may like

- AngularJS+Spring Security using Basic Authentication
- Secure Spring REST API using Basic Authentication

What is OAuth2

OAuth2 is an standardized authorization protocol/framework. As per Official OAuth2 Specification: The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

Big players like Google, Facebook and others are already using their own OAuth2 implementations for quite some time. Enterprises too are moving fast towards OAuth2 adoption.

I found OAuth2 specification rather simple to follow. Yet if you want to start even quickly, an excellent article on OAuth2 fundamentals can be found here which gives a deep insight in OAUth2 theoretical concepts.
Spring Security OAuth project provides all the necessary API we might need in order to develop an OAuth2 compliant implementation using Spring. Official Spring security oauth project provides a comprehensive example for implementing OAuth2. The code samples of this post is inspired by that examples itself. The intention of this post is to just use bare-minimum functionality required in order to secure our REST API, nothing more. As you, I too am still learning it, so feel free to correct me if something seems not right.
At minimum, you should be aware of four key concepts in OAuth2:

1. OAuth2 Roles

OAuth2 defines four roles:

- `resource owner`:
  Could be you. An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

- `resource server`:
  The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
- `client`:
  An application making protected resource requests on behalf of the resource owner and with its authorization. It could be a mobile app asking your permission to access your Facebook feeds, a REST client trying to access REST API, a web site providing an alternative login option using Facebook account.
- `authorization server`:
  The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

In our example, our REST API can only be accessed via Resource server which will require an access token to be present with request

2. OAuth2 Authorization Grant types

An authorization grant is a credential representing the resource owner's authorization (to access its protected resources) used by the client to obtain an access token. The specification defines four grant types:

- `authorization code`
- `implicit`
- `resource owner password credentials`
- `client credentials`

We will be using `resource owner password credentials` grant type. The reason is simple, we are not implementing a view which redirects us to a login page. Only the usage where a client [Postman or RestTemplate based Java client e.g.] have the Resource owner's credentials and they provide those credential [along with client credentials] to authorization server in order to eventually receive the access-token[and optionally refresh token], and then use that token to actually access the resources.

A common example is the `GMail app` [a client] on your smartphone which takes your credentials and use them to connect to `GMail servers`. It also shows that 'Password Credentials Grant' is best suited when both the client and the servers are from same company as the trust is there, you don't want to provide your credentials to a third party.

3. OAuth2 Tokens

Tokens are implementation specific random strings, generated by the authorization server and are issued when the client requests them.

- `Access Token` : Sent with each request, usually valid for a very short life time [an hour e.g.]
- `Refresh Token` : Mainly used to get a new access token, not sent with each request, usually lives longer than access token.

A Word on HTTPS : For any sort of Security implementation, ranging from Basic authentication to a full fledged OAuth2 implementation, `HTTPS` is a must have. Without HTTPS, no matter what your implementation is, security is vulnerable to be compromised.

4. OAuth2 Access Token Scope

Client can ask for the resource with specific access rights using scope [want to access feeds & photos of this users facebook account], and authorization server in turn return scope showing what access rights were actually granted to the client [Resource owner only allowed feeds access, no photos e.g.].

Let's Get into Code

Let's implement the necessary building blocks to implement OAuth using Spring Security, in order to access our REST resources.

1. Resource Server

Resource Server hosts the resources [our REST API] the client is interested in. Resources are located on `/user/`. `@EnableResourceServer` annotation, applied on OAuth2 Resource Servers, enables a Spring Security filter that authenticates requests using an incoming OAuth2 token.
Class `ResourceServerConfigurerAdapter`implements `ResourceServerConfigurer` providing methods to adjust the access rules and paths that are protected by OAuth2 security.

```
package com.websystique.springmvc.security;

import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Enab
leResourceServer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Reso
urceServerConfigurerAdapter;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Resour
ceServerSecurityConfigurer;
import
org.springframework.security.oauth2.provider.error.OAuth2AccessDeniedHandler
;

@Configuration
@EnableResourceServer
public class ResourceServerConfiguration extends
ResourceServerConfigurerAdapter {

    private static final String RESOURCE_ID = "my_rest_api";

    @Override
    public void configure(ResourceServerSecurityConfigurer resources) {
        resources.resourceId(RESOURCE_ID).stateless(false);
    }

    @Override
```

```
    public void configure(HttpSecurity http) throws Exception {
        http.
        anonymous().disable()
        .requestMatchers().antMatchers("/user/**")
        .and().authorizeRequests()
        .antMatchers("/user/**").access("hasRole('ADMIN')")
        .and().exceptionHandling().accessDeniedHandler(new
OAuth2AccessDeniedHandler());
    }

}
```

2. Authorization Server

Authorization server is the one responsible for verifying credentials and if credentials are OK, providing the tokens[refresh-token as well as access-token]. It also contains information about registered clients and possible access scopes and grant types. The token store is used to store the token. We will be using an in-memory token store.`@EnableAuthorizationServer` enables an Authorization Server (i.e. an AuthorizationEndpoint and a TokenEndpoint) in the current application context.

Class `AuthorizationServerConfigurerAdapter` implements `AuthorizationServerConfigurer` which provides all the necessary methods to configure an Authorization server.

```
package com.websystique.springmvc.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigure
r;
import
org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConf
igurerAdapter;
import
org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServ
er;
import
org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpoi
ntsConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecuri
tyConfigurer;
import org.springframework.security.oauth2.provider.approval.UserApprovalHandler;
import org.springframework.security.oauth2.provider.token.TokenStore;

@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfiguration extends AuthorizationServerConfigurerAdapter {

    private static String REALM="MY_OAUTH_REALM";

    @Autowired
    private TokenStore tokenStore;

    @Autowired
    private UserApprovalHandler userApprovalHandler;

    @Autowired
    @Qualifier("authenticationManagerBean")
    private AuthenticationManager authenticationManager;

    @Override
```

```
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

        clients.inMemory()
            .withClient("my-trusted-client")
            .authorizedGrantTypes("password", "authorization_code", "refresh_token",
"implicit")
            .authorities("ROLE_CLIENT", "ROLE_TRUSTED_CLIENT")
            .scopes("read", "write", "trust")
            .secret("secret")
            .accessTokenValiditySeconds(120).//Access token is only valid for 2 minutes.
            refreshTokenValiditySeconds(600);//Refresh token is only valid for 10 minutes.
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints.tokenStore(tokenStore).userApprovalHandler(userApprovalHandler)
                .authenticationManager(authenticationManager);
    }

    @Override
    public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {
        oauthServer.realm(REALM+"/client");
    }

}
```

Above configuration

- Registers a client with client-id 'my-trusted-client' and password 'secret' and roles & scope he is allowed for.
- Specifies that any generated access token will be valid for only 120 seconds
- Specifies that any generated refresh token will be valid for only 600 seconds

3. Security Configuration

Gluing everything together. Endpoint `/oauth/token` is used to request a token [access or refresh]. Resource owners [bill,bob] are configured here itself.

```
package com.websystique.springmvc.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.Authe
nticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSe
curity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurity
ConfigurerAdapter;
import org.springframework.security.oauth2.provider.ClientDetailsService;
import org.springframework.security.oauth2.provider.approval.ApprovalStore;
import
org.springframework.security.oauth2.provider.approval.TokenApprovalStore;
```

```
import
org.springframework.security.oauth2.provider.approval.TokenStoreUserApproval
Handler;
import
org.springframework.security.oauth2.provider.request.DefaultOAuth2RequestFac
tory;
import org.springframework.security.oauth2.provider.token.TokenStore;
import
org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;

@Configuration
@EnableWebSecurity
public class OAuth2SecurityConfiguration extends WebSecurityConfigurerAdapter
{

    @Autowired
    private ClientDetailsService clientDetailsService;

    @Autowired
    public void globalUserDetails(AuthenticationManagerBuilder auth) throws
Exception {
        auth.inMemoryAuthentication()
        .withUser("bill").password("abc123").roles("ADMIN").and()
        .withUser("bob").password("abc123").roles("USER");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
        .csrf().disable()
        .anonymous().disable()
        .authorizeRequests()
        .antMatchers("/oauth/token").permitAll();
    }

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception
{
        return super.authenticationManagerBean();
    }



    @Bean
    public TokenStore tokenStore() {
        return new InMemoryTokenStore();
    }

    @Bean
    @Autowired
    public TokenStoreUserApprovalHandler userApprovalHandler(TokenStore
tokenStore){
        TokenStoreUserApprovalHandler handler = new
TokenStoreUserApprovalHandler();
```

```
        handler.setTokenStore(tokenStore);
        handler.setRequestFactory(new
DefaultOAuth2RequestFactory(clientDetailsService));
        handler.setClientDetailsService(clientDetailsService);
        return handler;
    }

    @Bean
    @Autowired
    public ApprovalStore approvalStore(TokenStore tokenStore) throws
Exception {
        TokenApprovalStore store = new TokenApprovalStore();
        store.setTokenStore(tokenStore);
        return store;
    }

}
```

Additionally, enable Global method security which will activate @PreFilter, @PostFilter, @PreAuthorize @PostAuthorize annotations if we want to use them.

```
package com.websystique.springmvc.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.access.expression.method.MethodSecurityExpressi
onHandler;
import
org.springframework.security.config.annotation.method.configuration.EnableGl
obalMethodSecurity;
import
org.springframework.security.config.annotation.method.configuration.GlobalMe
thodSecurityConfiguration;
import
org.springframework.security.oauth2.provider.expression.OAuth2MethodSecurity
ExpressionHandler;

@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true, proxyTargetClass = true)
public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {
    @Autowired
    private OAuth2SecurityConfiguration securityConfig;

    @Override
    protected MethodSecurityExpressionHandler createExpressionHandler() {
        return new OAuth2MethodSecurityExpressionHandler();
    }
}
```

4. Endpoints and their purpose

- Attempt to access resources [REST API] without any authorization [will fail of-course].
  `GET http://localhost:8080/SpringSecurityOAuth2Example/user/`

- Ask for tokens[access+refresh] using **HTTP POST** on `/oauth/token`, with grant_type=password,and resource owners credentials as req-params. Additionally, send client credentials in Authorization header.
  `POST http://localhost:8080/SpringSecurityOAuth2Example/oauth/token?grant_type=password&username=bill&password=abc123`
- Ask for a new access token via valid refresh-token, using **HTTP POST** on `/oauth/token`, with grant_type=refresh_token,and sending actual refresh token. Additionally, send client credentials in Authorization header.
  `POST http://localhost:8080/SpringSecurityOAuth2Example/oauth/token?grant_type=refresh_token&refresh_token=094b7d23-973f-4cc1-83ad-8ffd43de1845`
- Access the resource by providing an access token using `access_token` query param with request.
  `GET http://localhost:8080/SpringSecurityOAuth2Example/user/?access_token=3525d0e4-d881-49e7-9f91-bcfd18259109`

5. Rest API

The simple Spring REST API i used in most of my posts.

```
package com.websystique.springmvc.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.util.UriComponentsBuilder;

import com.websystique.springmvc.model.User;
import com.websystique.springmvc.service.UserService;

@RestController
public class HelloWorldRestController {

    @Autowired
    UserService userService;  //Service which will do all data
retrieval/manipulation work
```

```
    //-------------------Retrieve All Users--------------------------------
-----------------------

    @RequestMapping(value = "/user/", method = RequestMethod.GET)
    public ResponseEntity<List<User>> listAllUsers() {
        List<User> users = userService.findAllUsers();
        if(users.isEmpty()){
            return new
ResponseEntity<List<User>>(HttpStatus.NO_CONTENT);//You many decide to
return HttpStatus.NOT_FOUND
        }
        return new ResponseEntity<List<User>>(users, HttpStatus.OK);
    }


    //-------------------Retrieve Single User------------------------------
-------------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET,
produces =
{MediaType.APPLICATION_JSON_VALUE,MediaType.APPLICATION_XML_VALUE})
    public ResponseEntity<User> getUser(@PathVariable("id") long id) {
        System.out.println("Fetching User with id " + id);
        User user = userService.findById(id);
        if (user == null) {
            System.out.println("User with id " + id + " not found");
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<User>(user, HttpStatus.OK);
    }



    //-------------------Create a User-------------------------------------
------------------

    @RequestMapping(value = "/user/", method = RequestMethod.POST)
    public ResponseEntity<Void> createUser(@RequestBody User user,
UriComponentsBuilder ucBuilder) {
        System.out.println("Creating User " + user.getName());

        if (userService.isUserExist(user)) {
            System.out.println("A User with name " + user.getName() + "
already exist");
            return new ResponseEntity<Void>(HttpStatus.CONFLICT);
        }

        userService.saveUser(user);

        HttpHeaders headers = new HttpHeaders();
        headers.setLocation(ucBuilder.path("/user/{id}").buildAndExpand(user
.getId()).toUri());
        return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
```

```
    }


    //------------------- Update a User ------------------------------------
--------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.PUT)
    public ResponseEntity<User> updateUser(@PathVariable("id") long id,
@RequestBody User user) {
        System.out.println("Updating User " + id);

        User currentUser = userService.findById(id);

        if (currentUser==null) {
            System.out.println("User with id " + id + " not found");
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
        }

        currentUser.setName(user.getName());
        currentUser.setAge(user.getAge());
        currentUser.setSalary(user.getSalary());

        userService.updateUser(currentUser);
        return new ResponseEntity<User>(currentUser, HttpStatus.OK);
    }

    //------------------- Delete a User ------------------------------------
--------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<User> deleteUser(@PathVariable("id") long id) {
        System.out.println("Fetching & Deleting User with id " + id);

        User user = userService.findById(id);
        if (user == null) {
            System.out.println("Unable to delete. User with id " + id + "
not found");
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
        }

        userService.deleteUserById(id);
        return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
    }


    //------------------- Delete All Users ---------------------------------
-----------------------

    @RequestMapping(value = "/user/", method = RequestMethod.DELETE)
    public ResponseEntity<User> deleteAllUsers() {
        System.out.println("Deleting All Users");
```

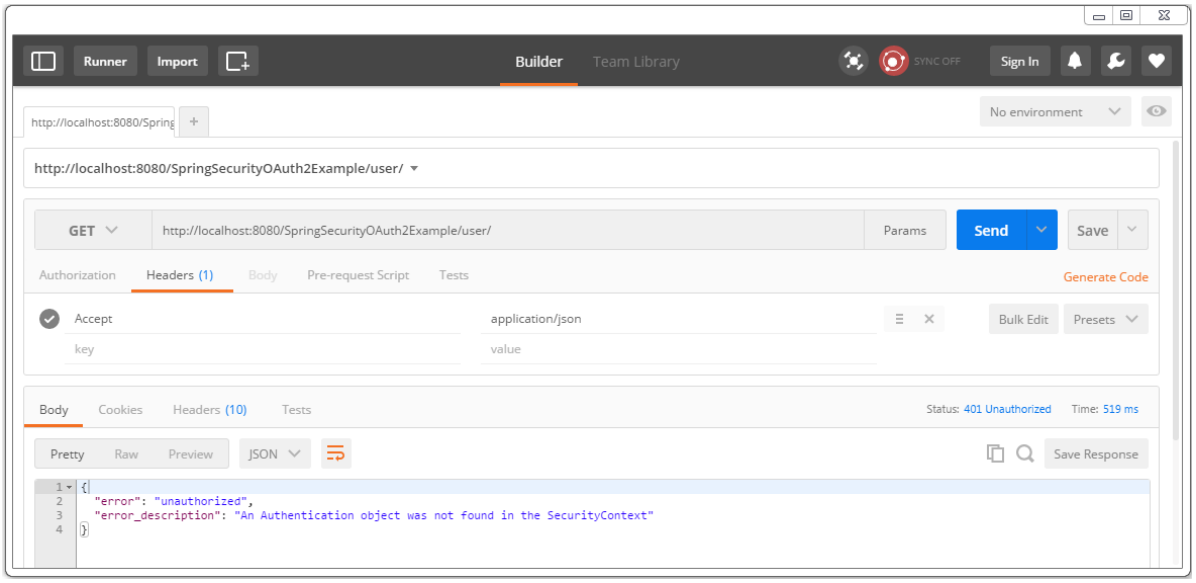
```
        userService.deleteAllUsers();
        return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
    }

}
```

6. Running the application

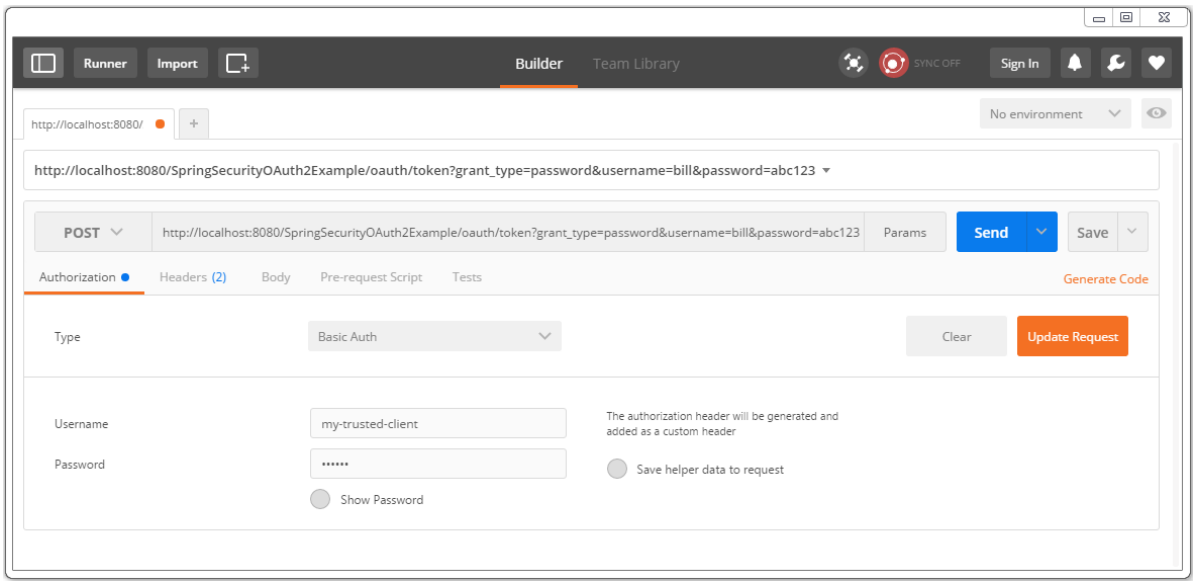
Run it and test it using two different clients.

Client 1: Postman

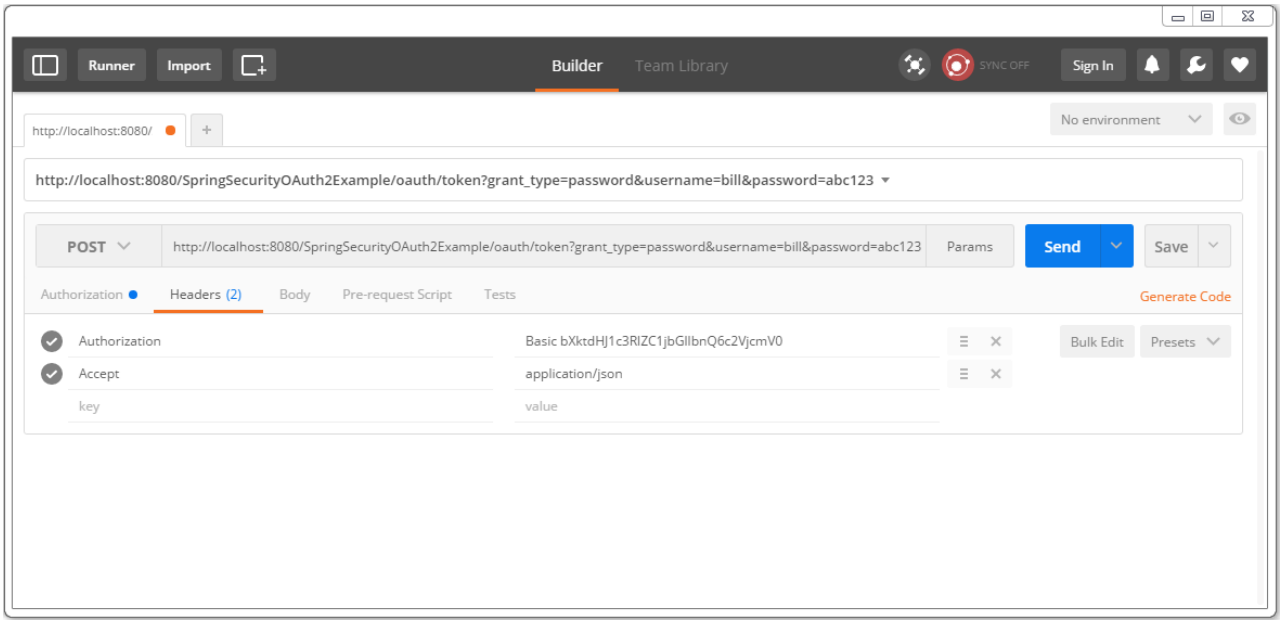
Try to access a resource without any auth info, wil get a 401.



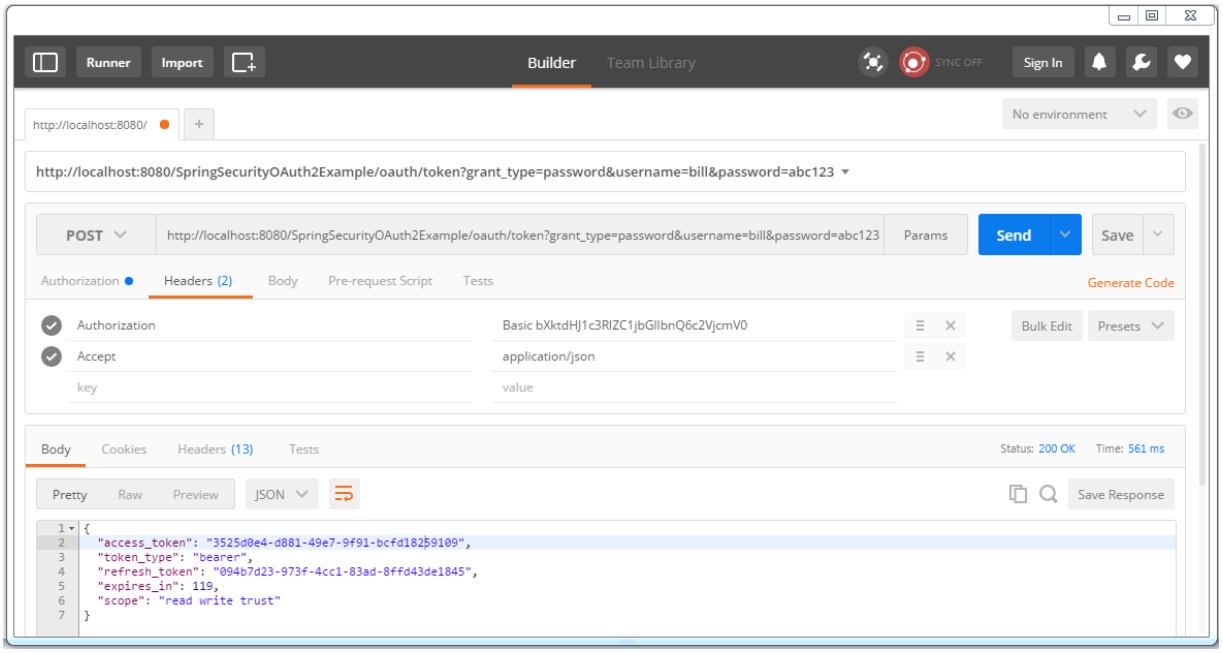
Let's get the tokens. First add an authorization header with **client credentials** [my-trusted-client/secret].



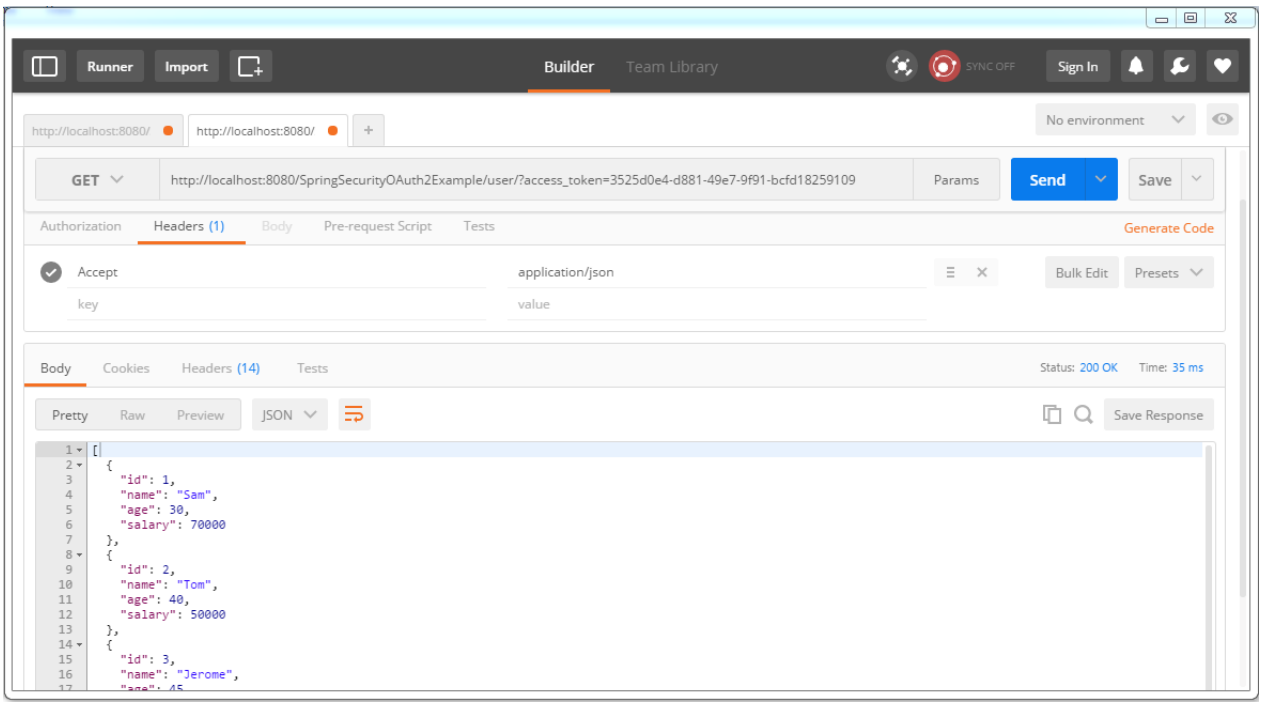
Click on update request, verify the header in header-tab.



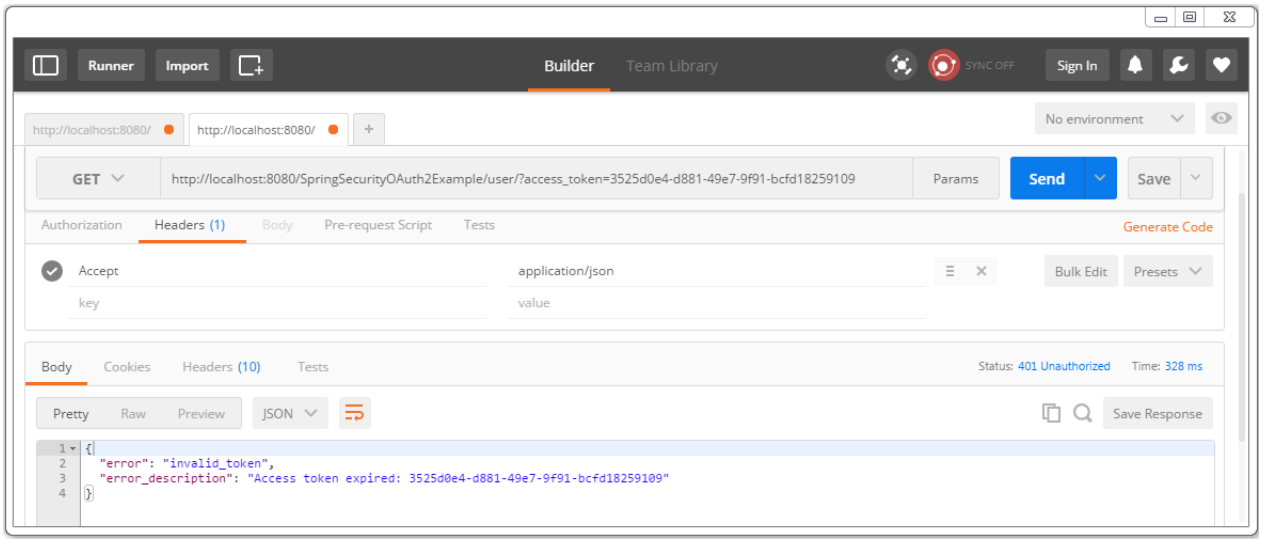
Send the Post request, you should receive the response containing `access-token` as well as `refresh-token`.



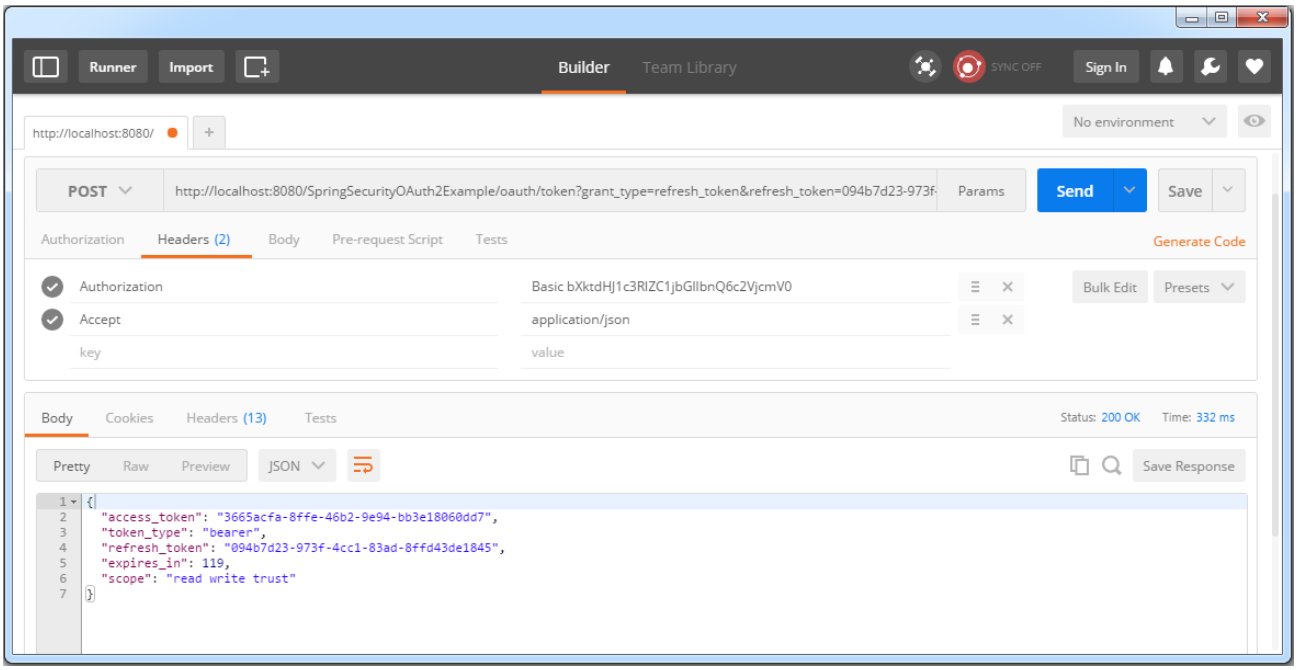
Save these tokens somewhere, you will need them. Now you can use this access-token [valid for 2 minutes] to access resources.



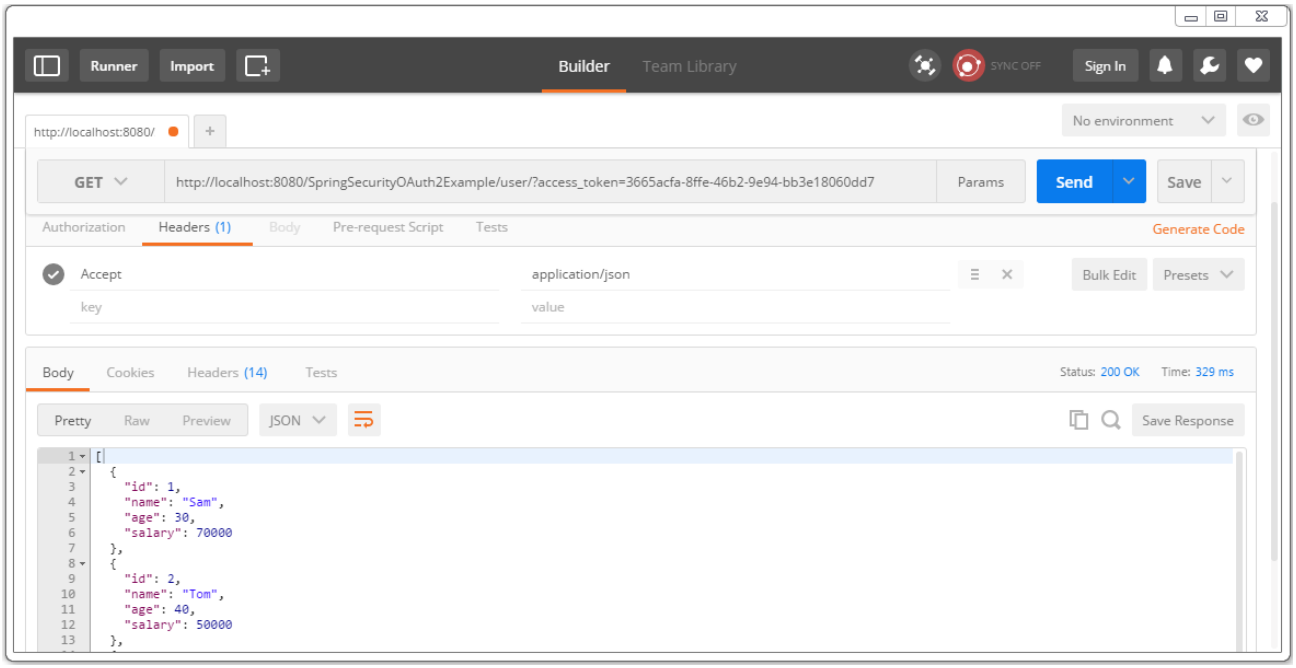
After 2 minutes, access-token gets expired, your further resource requests will fail.



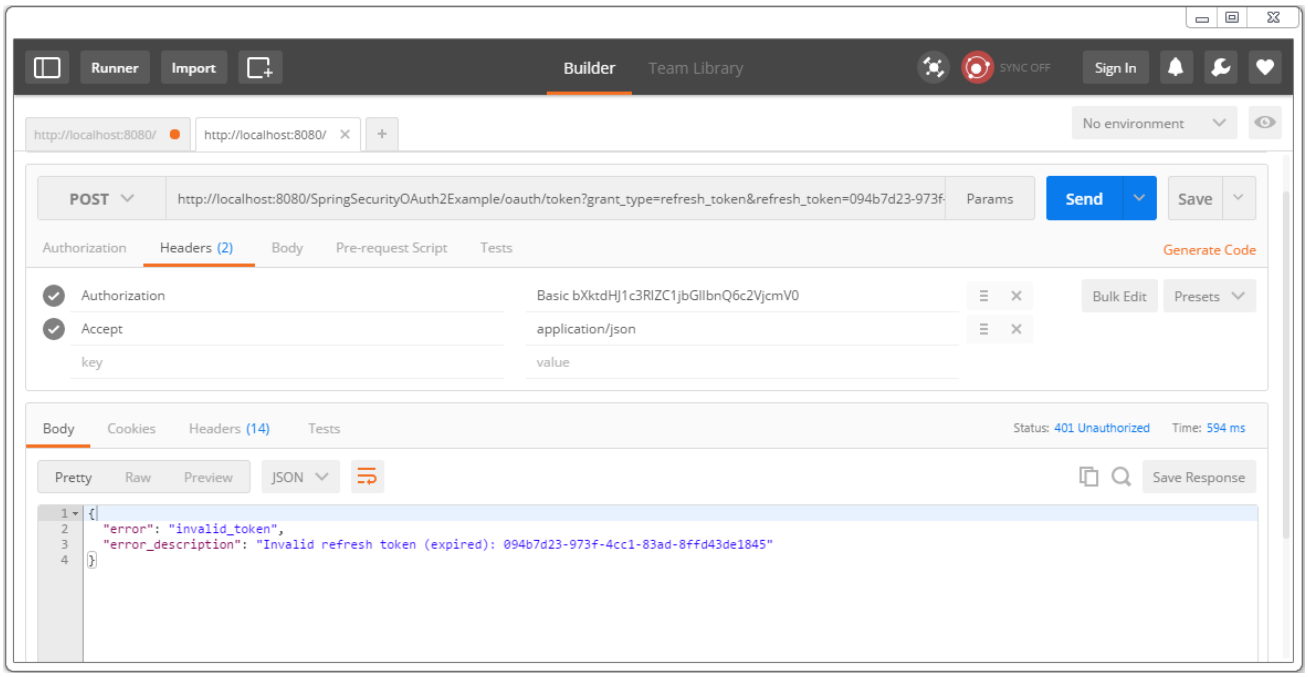
We need a new access-token. Fire a post to with refresh-token to get a brand-new access-token.



Use this new access-token to access the resources.



Refresh-token expires too[10 minutes]. After that, you should see your refresh request getting failed.



It means you need to request a new refresh+access-token, as in step 2.

Client 2: RestTemplate based java application

Method **sendTokenRequest** is used to actually get the tokens. The access-token we got in response is then used with each request. If required, You can implement the refresh-token flow easily in below example.

```
package com.websystique.springmvc;

import java.net.URI;
import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.List;

import org.apache.commons.codec.binary.Base64;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.util.Assert;
import org.springframework.web.client.RestTemplate;

import com.websystique.springmvc.model.AuthTokenInfo;
import com.websystique.springmvc.model.User;

public class SpringRestClient {

    public static final String REST_SERVICE_URI =
"http://localhost:8080/SpringSecurityOAuth2Example";

    public static final String AUTH_SERVER_URI =
"http://localhost:8080/SpringSecurityOAuth2Example/oauth/token";

    public static final String QPM_PASSWORD_GRANT =
"?grant_type=password&username=bill&password=abc123";

    public static final String QPM_ACCESS_TOKEN = "?access_token=";

    /*
     * Prepare HTTP Headers.
     */
    private static HttpHeaders getHeaders(){
        HttpHeaders headers = new HttpHeaders();
        headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
        return headers;
    }

    /*
     * Add HTTP Authorization header, using Basic-Authentication to send
client-credentials.
     */
    private static HttpHeaders getHeadersWithClientCredentials(){
        String plainClientCredentials="my-trusted-client:secret";
        String base64ClientCredentials = new
String(Base64.encodeBase64(plainClientCredentials.getBytes()));
```

```
        HttpHeaders headers = getHeaders();
        headers.add("Authorization", "Basic " + base64ClientCredentials);
        return headers;
    }

    /*
     * Send a POST request [on /oauth/token] to get an access-token, which
will then be send with each request.
     */
    @SuppressWarnings({ "unchecked"})
    private static AuthTokenInfo sendTokenRequest(){
        RestTemplate restTemplate = new RestTemplate();

        HttpEntity<String> request = new
HttpEntity<String>(getHeadersWithClientCredentials());
        ResponseEntity<Object> response =
restTemplate.exchange(AUTH_SERVER_URI+QPM_PASSWORD_GRANT, HttpMethod.POST,
request, Object.class);
        LinkedHashMap<String, Object> map = (LinkedHashMap<String,
Object>)response.getBody();
        AuthTokenInfo tokenInfo = null;

        if(map!=null){
            tokenInfo = new AuthTokenInfo();
            tokenInfo.setAccess_token((String)map.get("access_token"));
            tokenInfo.setToken_type((String)map.get("token_type"));
            tokenInfo.setRefresh_token((String)map.get("refresh_token"));
            tokenInfo.setExpires_in((int)map.get("expires_in"));
            tokenInfo.setScope((String)map.get("scope"));
            System.out.println(tokenInfo);
            //System.out.println("access_token ="+map.get("access_token")+",
token_type="+map.get("token_type")+",
refresh_token="+map.get("refresh_token")
            //+", expires_in="+map.get("expires_in")+",
scope="+map.get("scope"));;
        }else{
            System.out.println("No user exist----------");

        }
        return tokenInfo;
    }

    /*
     * Send a GET request to get list of all users.
     */
    @SuppressWarnings({ "unchecked", "rawtypes" })
    private static void listAllUsers(AuthTokenInfo tokenInfo){
        Assert.notNull(tokenInfo, "Authenticate first please.....");

        System.out.println("\nTesting listAllUsers API-----------");
        RestTemplate restTemplate = new RestTemplate();
```

```
        HttpEntity<String> request = new HttpEntity<String>(getHeaders());
        ResponseEntity<List> response =
restTemplate.exchange(REST_SERVICE_URI+"/user/"+QPM_ACCESS_TOKEN+tokenInfo.g
etAccess_token(),
                HttpMethod.GET, request, List.class);
        List<LinkedHashMap<String, Object>> usersMap =
(List<LinkedHashMap<String, Object>>)response.getBody();

        if(usersMap!=null){
            for(LinkedHashMap<String, Object> map : usersMap){
                System.out.println("User : id="+map.get("id")+",
Name="+map.get("name")+", Age="+map.get("age")+",
Salary="+map.get("salary"));;
            }
        }else{
            System.out.println("No user exist----------");
        }
    }

    /*
     * Send a GET request to get a specific user.
     */
    private static void getUser(AuthTokenInfo tokenInfo){
        Assert.notNull(tokenInfo, "Authenticate first please......");
        System.out.println("\nTesting getUser API----------");
        RestTemplate restTemplate = new RestTemplate();
        HttpEntity<String> request = new HttpEntity<String>(getHeaders());
        ResponseEntity<User> response =
restTemplate.exchange(REST_SERVICE_URI+"/user/1"+QPM_ACCESS_TOKEN+tokenInfo.
getAccess_token(),
                HttpMethod.GET, request, User.class);
        User user = response.getBody();
        System.out.println(user);
    }

    /*
     * Send a POST request to create a new user.
     */
    private static void createUser(AuthTokenInfo tokenInfo) {
        Assert.notNull(tokenInfo, "Authenticate first please......");
        System.out.println("\nTesting create User API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user = new User(0,"Sarah",51,134);
        HttpEntity<Object> request = new HttpEntity<Object>(user,
getHeaders());
        URI uri =
restTemplate.postForLocation(REST_SERVICE_URI+"/user/"+QPM_ACCESS_TOKEN+toke
nInfo.getAccess_token(),
                request, User.class);
        System.out.println("Location : "+uri.toASCIIString());
    }

    /*
     * Send a PUT request to update an existing user.
```

```
     */
    private static void updateUser(AuthTokenInfo tokenInfo) {
        Assert.notNull(tokenInfo, "Authenticate first please......");
        System.out.println("\nTesting update User API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user  = new User(1,"Tomy",33, 70000);
        HttpEntity<Object> request = new HttpEntity<Object>(user,
getHeaders());
        ResponseEntity<User> response =
restTemplate.exchange(REST_SERVICE_URI+"/user/1"+QPM_ACCESS_TOKEN+tokenInfo.
getAccess_token(),
                HttpMethod.PUT, request, User.class);
        System.out.println(response.getBody());
    }

    /*
     * Send a DELETE request to delete a specific user.
     */
    private static void deleteUser(AuthTokenInfo tokenInfo) {
        Assert.notNull(tokenInfo, "Authenticate first please......");
        System.out.println("\nTesting delete User API----------");
        RestTemplate restTemplate = new RestTemplate();
        HttpEntity<String> request = new HttpEntity<String>(getHeaders());
        restTemplate.exchange(REST_SERVICE_URI+"/user/3"+QPM_ACCESS_TOKEN+to
kenInfo.getAccess_token(),
                HttpMethod.DELETE, request, User.class);
    }



    /*
     * Send a DELETE request to delete all users.
     */
    private static void deleteAllUsers(AuthTokenInfo tokenInfo) {
        Assert.notNull(tokenInfo, "Authenticate first please......");
        System.out.println("\nTesting all delete Users API----------");
        RestTemplate restTemplate = new RestTemplate();
        HttpEntity<String> request = new HttpEntity<String>(getHeaders());
        restTemplate.exchange(REST_SERVICE_URI+"/user/"+QPM_ACCESS_TOKEN+tok
enInfo.getAccess_token(),
                HttpMethod.DELETE, request, User.class);
    }

    public static void main(String args[]){
        AuthTokenInfo tokenInfo = sendTokenRequest();
        listAllUsers(tokenInfo);

        getUser(tokenInfo);

        createUser(tokenInfo);
        listAllUsers(tokenInfo);

        updateUser(tokenInfo);
        listAllUsers(tokenInfo);
```

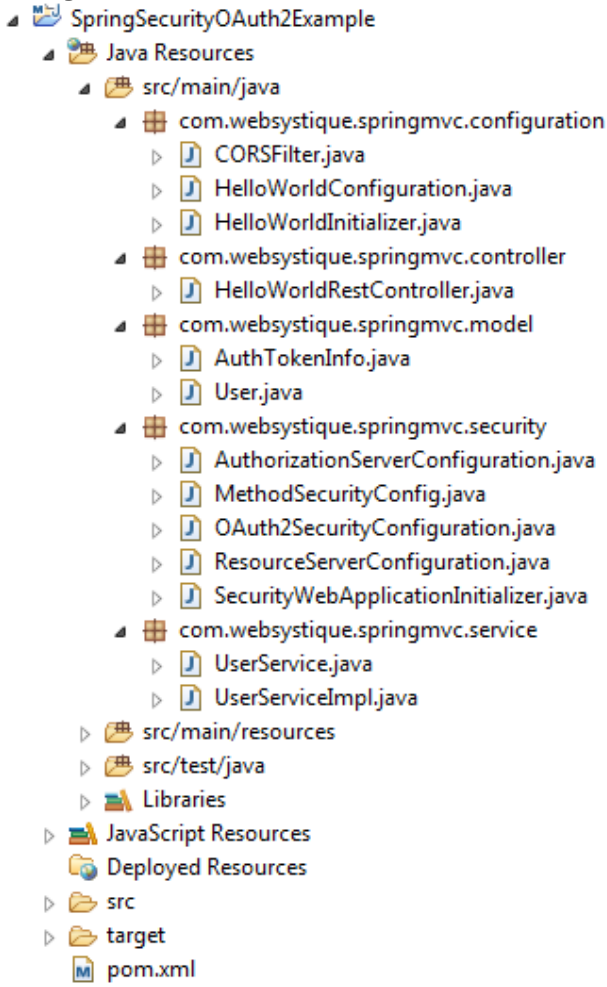
```
        deleteUser(tokenInfo);
        listAllUsers(tokenInfo);

        deleteAllUsers(tokenInfo);
        listAllUsers(tokenInfo);
    }
}
```

Above code will produce following output:

```
AuthTokenInfo [access_token=fceed386-5923-4bf8-b193-1d76f95da4c4,
token_type=bearer, refresh_token=29d28ee2-9d09-483f-a2d6-7f93e7a31667,
expires_in=71, scope=read write trust]

Testing listAllUsers API-----------
User : id=1, Name=Sam, Age=30, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0

Testing getUser API----------
User [id=1, name=Sam, age=30, salary=70000.0]

Testing create User API----------
Location : http://localhost:8080/SpringSecurityOAuth2Example/user/5

Testing listAllUsers API-----------
User : id=1, Name=Sam, Age=30, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0

Testing update User API----------
User [id=1, name=Tomy, age=33, salary=70000.0]

Testing listAllUsers API-----------
User : id=1, Name=Tomy, Age=33, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0

Testing delete User API----------

Testing listAllUsers API-----------
User : id=1, Name=Tomy, Age=33, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0

Testing all delete Users API----------
```

```
Testing listAllUsers API-----------
No user exist----------
```

Project Structure



pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.websystique.springmvc</groupId>
    <artifactId>SpringSecurityOAuth2Example</artifactId>
    <version>1.0.0</version>
    <packaging>war</packaging>

    <name>SpringSecurityOAuth2Example</name>

    <properties>
```

```
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <springframework.version>4.3.1.RELEASE</springframework.version>
        <springsecurity.version>4.1.1.RELEASE</springsecurity.version>
        <springsecurityoauth2.version>2.0.10.RELEASE</springsecurityoauth2.v
ersion>
        <jackson.library>2.7.5</jackson.library>
    </properties>

    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${springframework.version}</version>
        </dependency>

        <!-- Spring Security -->
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${springsecurity.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${springsecurity.version}</version>
        </dependency>

        <!-- Spring Security OAuth2-->
        <dependency>
            <groupId>org.springframework.security.oauth</groupId>
            <artifactId>spring-security-oauth2</artifactId>
            <version>${springsecurityoauth2.version}</version>
        </dependency>

        <!-- Jackson libraries -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.library}</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
```

```
            <version>${jackson.library}</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
        </dependency>
    </dependencies>

    <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.2</version>
                    <configuration>
                        <source>1.7</source>
                        <target>1.7</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>2.4</version>
                    <configuration>
                        <warSourceDirectory>src/main/webapp</warSourceDirect
ory>
                        <warName>SpringSecurityOAuth2Example</warName>
                        <failOnMissingWebXml>false</failOnMissingWebXml>
                    </configuration>
                </plugin>
            </plugins>
        <finalName>SpringSecurityOAuth2Example</finalName>
    </build>
</project>
```

Download Source Code

Download Now!

References

- OAuth2 Specification
- Spring OAuth2 Official reference
- Spring Security 4 Project Page
- Spring Security 4 Reference Manual