Secure Spring REST API using Basic Authentication

Created on: July 30, 2016 | **Last updated on:** November 14, 2016 websystiqueadmin

So You've got the REST API for your application, and now you want to secure it. How to do that? There are several popular ways to do that, ranging from `Basic Authentication` to a full fledged `OAuth2` security solution. This Guide explains securing REST API using Basic Authentication with help of examples involving two separate clients [Postman & a `Spring RestTemplate` based Java app] trying to get access to our REST API. We will be showing the same example with OAuth2 in the next post Secure REST API using OAuth2.
As always, complete code can be found in attachment at the end of this article.

In case you are looking for AngularJS based app using Basic Authentication, Post AngularJS BasicAuthentication using Spring Security shows how this application can be used with an AngularJS client.

What is Basic Authentication?

Traditional authentication approaches like login pages or session identification are good for web based clients involving human interaction but does not really fit well when communicating with [REST] clients which may not even be a web application. Think of an API over a server which tries to communicate with another API on a totally different server, without any human intervention.

Basic Authentication provides a solution for this problem, although not very secure. With Basic Authentication, clients send it's Base64 encoded credentials **with each request**, using HTTP [Authorization] header . That means each request is independent of other request and server may/does not maintain any state information for the client, which is good for scalability point of view.

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.

Shown below is the sample code for preparing the header.

```
String plainClientCredentials="myusername:mypassword";
String base64ClientCredentials = new
String(Base64.encodeBase64(plainClientCredentials.getBytes()));

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

which may in turn produce something like:
`Authorization : Basic bXktdHJ1c3RlZC1jbGllbnQ6c2VjcmV0...`

This header will be sent with ech request. Since Credentials [Base 64 encoded, not even encrypted] are sent with each request, they can be compromised. One way to prevent this is **using HTTPS** in conjunction with Basic Authentication.

Basic Authentication & Spring Security

With two steps, you can enable the Basic Authentication in Spring Security Configuration.

1. `Configure httpBasic` : Configures HTTP Basic authentication. [**http-basic** in XML]
2. `Configure authentication entry point with BasicAuthenticationEntryPoint` : In case the Authentication fails [invalid/missing credentials], this entry point will get triggered. It is very important, because we don't want [Spring Security default behavior] of redirecting to a login page on authentication failure [We don't have a login page].

Shown below is the complete Spring Security configuration with httpBasic and entry point setup.

```
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.http.HttpMethod;
import
org.springframework.security.config.annotation.authentication.builders.Authen
ticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.builders.WebSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityC
onfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    private static String REALM="MY_TEST_REALM";

    @Autowired
    public void configureGlobalSecurity(AuthenticationManagerBuilder auth)
throws Exception {
        auth.inMemoryAuthentication().withUser("bill").password("abc123").rol
es("ADMIN");
        auth.inMemoryAuthentication().withUser("tom").password("abc123").role
s("USER");
    }

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

```
      http.csrf().disable()
        .authorizeRequests()
        .antMatchers("/user/**").hasRole("ADMIN")
        .and().httpBasic().realmName(REALM).authenticationEntryPoint(getBasic
AuthEntryPoint())
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolic
y.STATELESS);//We don't need sessions to be created.
    }

    @Bean
    public CustomBasicAuthenticationEntryPoint getBasicAuthEntryPoint(){
        return new CustomBasicAuthenticationEntryPoint();
    }

    /* To allow Pre-flight [OPTIONS] request from browser */
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers(HttpMethod.OPTIONS, "/**");
    }
}
```

And the actual Entry point, which will get triggerd if authentication failed. You can customize it to send custom content in response.

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

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;
import
org.springframework.security.web.authentication.www.BasicAuthenticationEntryP
oint;

public class CustomBasicAuthenticationEntryPoint extends
BasicAuthenticationEntryPoint {

    @Override
    public void commence(final HttpServletRequest request,
            final HttpServletResponse response,
            final AuthenticationException authException) throws IOException,
ServletException {
        //Authentication failed, send error response.
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        response.addHeader("WWW-Authenticate", "Basic realm=" + getRealmName()
+ "");

        PrintWriter writer = response.getWriter();
```

```
        writer.println("HTTP Status 401 : " + authException.getMessage());
    }

    @Override
    public void afterPropertiesSet() throws Exception {
        setRealmName("MY_TEST_REALM");
        super.afterPropertiesSet();
    }
}
```

That's all you need to configure basic security. Now let's see everything in action, with our good old REST API

REST API

Simple Spring REST API, which serves user(s). A client can perform CRUD operations using Standard HTML verbs, compliant with REST style.

```
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,
```

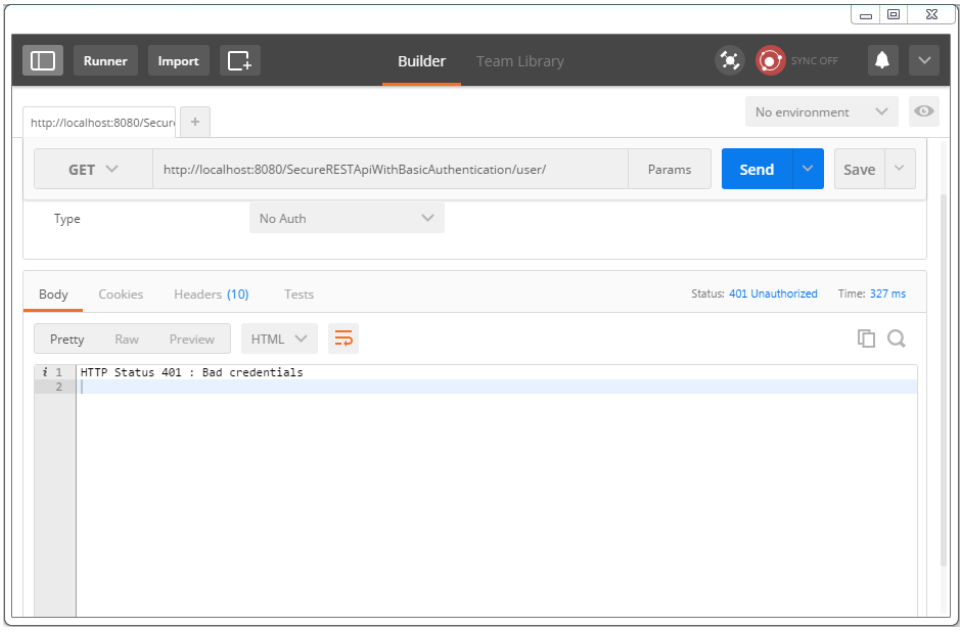
```
@RequestBody User user) {
        System.out.println("Updating User " + id);

        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);
    }

}
```

Running the application

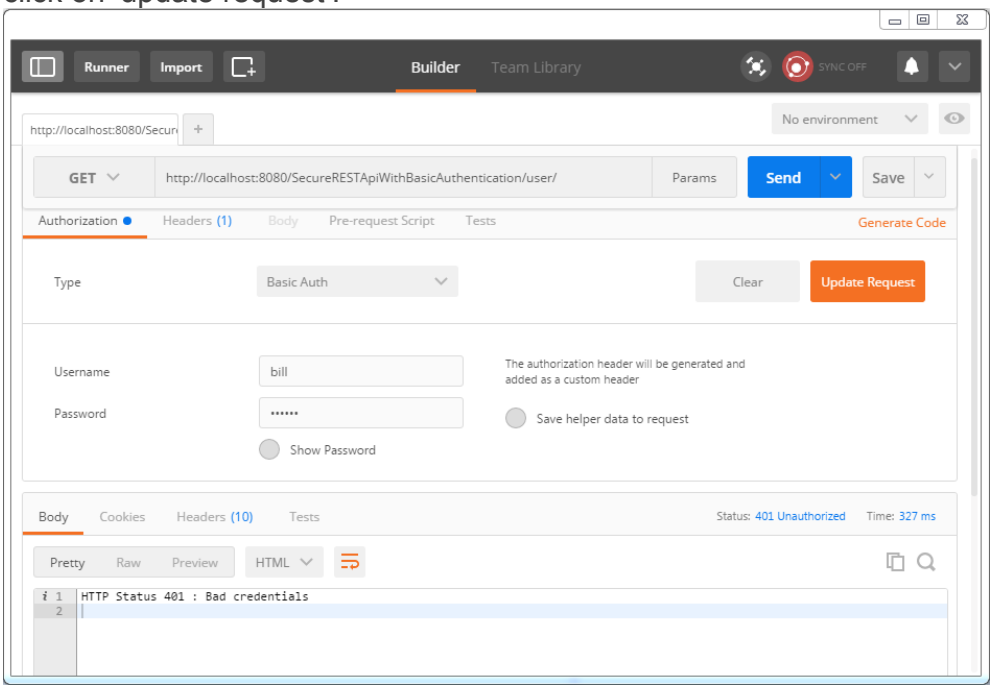
Build and deploy the application [on tomcat e.g]. Run it and test it using two different clients.

Using Client 1: Postman

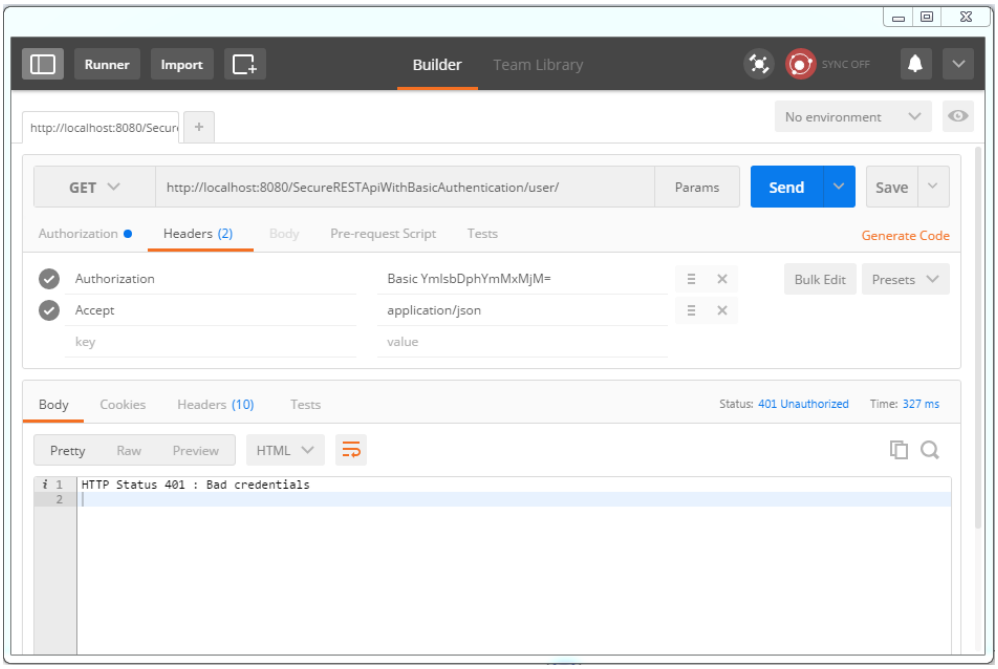
Send a request to get the list of users. You would get a 401 instead.



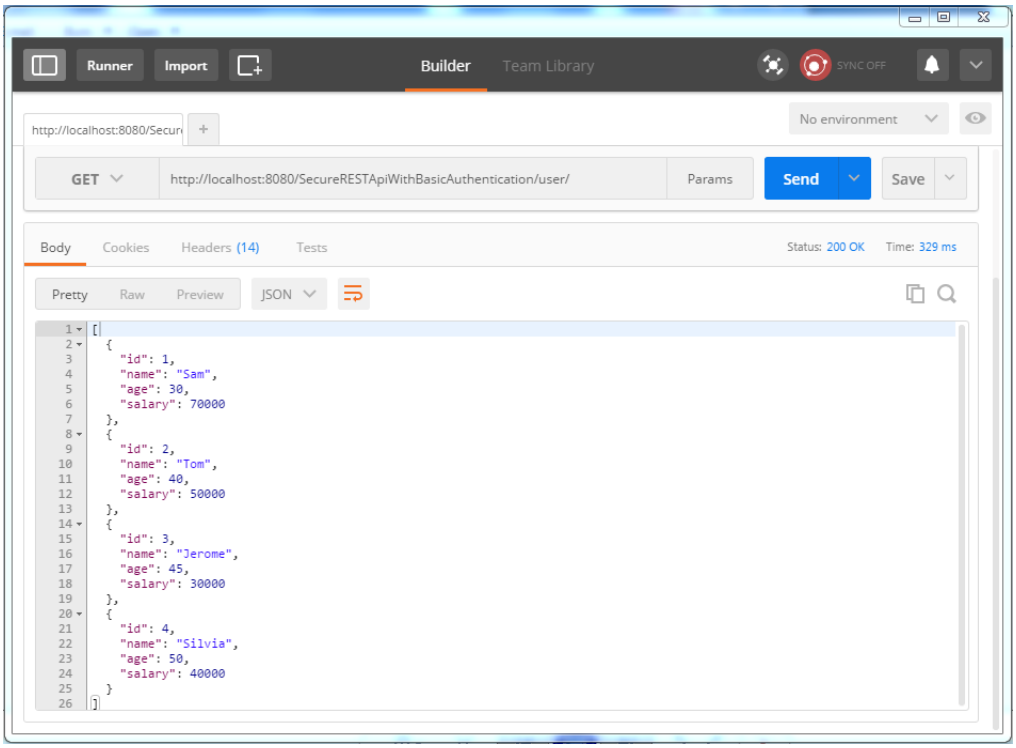
Now select type as ‘Basic Auth’ from dropdown, fill in username/password [bill/abc123], click on ‘update request’.



Click on Headers tab. You should see the new header. Let's add 'accept' header as well to enforce json response.



Now send the request. You should see the list of users in response this time.



Using Client 2: RestTemplate based Java Application

Let's use a full fledged Java client to access our REST API. We will be sending request using `Spring RestTemplate`. Take special note about how we are setting up the headers for each request, before sending the request.

```
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.web.client.RestTemplate;

import com.websystique.springmvc.model.User;

public class SpringRestClient {

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

    /*
     * Add HTTP Authorization header, using Basic-Authentication to send
user-credentials.
     */
    private static HttpHeaders getHeaders(){
        String plainCredentials="bill:abc123";
        String base64Credentials = new
String(Base64.encodeBase64(plainCredentials.getBytes()));

        HttpHeaders headers = new HttpHeaders();
        headers.add("Authorization", "Basic " + base64Credentials);
        headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
        return headers;
    }

    /*
     * Send a GET request to get list of all users.
     */
    @SuppressWarnings("unchecked")
    private static void listAllUsers(){
        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/", 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(){
        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", 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() {
        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/",
request, User.class);
        System.out.println("Location : "+uri.toASCIIString());
    }

    /*
     * Send a PUT request to update an existing user.
     */
    private static void updateUser() {
        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", HttpMethod.PUT, request,
```

```
User.class);
        System.out.println(response.getBody());
    }

    /*
     * Send a DELETE request to delete a specific user.
     */
    private static void deleteUser() {
        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", HttpMethod.DELETE,
request, User.class);
    }


    /*
     * Send a DELETE request to delete all users.
     */
    private static void deleteAllUsers() {
        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/", HttpMethod.DELETE,
request, User.class);
    }


    public static void main(String args[]){

        listAllUsers();

        getUser();

        createUser();
        listAllUsers();

        updateUser();
        listAllUsers();

        deleteUser();
        listAllUsers();

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

And the output is :

```
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/SecureRESTApiWithBasicAuthentication/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----------
```

Service being used in this example is shown below. The complete code can be found in attachment.

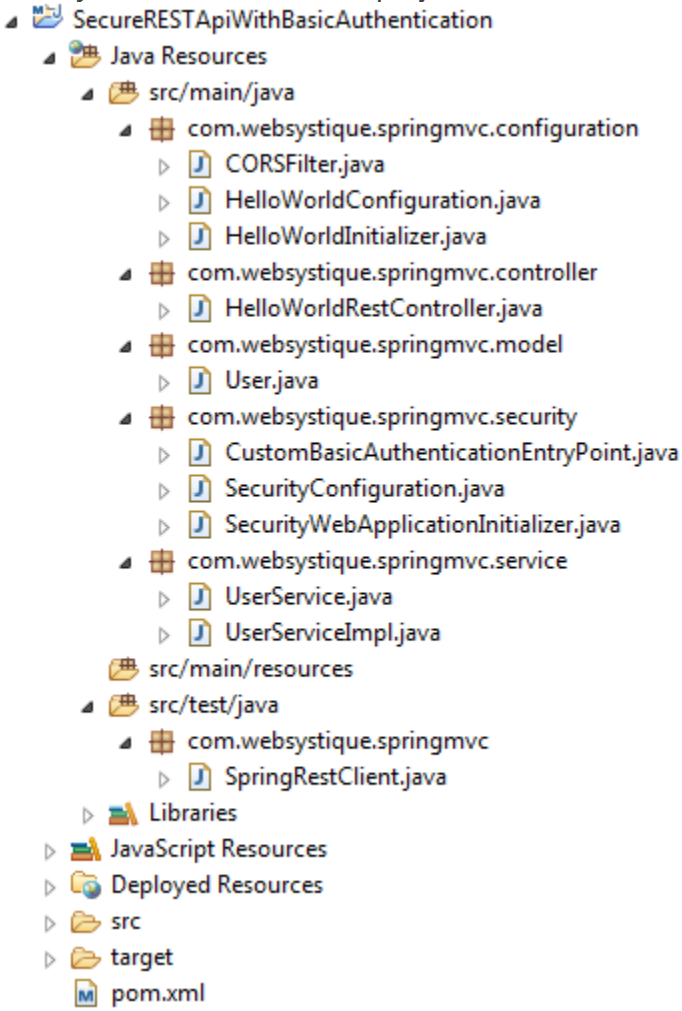
```
package com.websystique.springmvc.service;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.atomic.AtomicLong;

import org.springframework.stereotype.Service;

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

```
@Service("userService")
public class UserServiceImpl implements UserService{

    private static final AtomicLong counter = new AtomicLong();

    private static List<User> users;

    static{
        users= populateDummyUsers();
    }

    public List<User> findAllUsers() {
        return users;
    }

    public User findById(long id) {
        for(User user : users){
            if(user.getId() == id){
                return user;
            }
        }
        return null;
    }

    public User findByName(String name) {
        for(User user : users){
            if(user.getName().equalsIgnoreCase(name)){
                return user;
            }
        }
        return null;
    }

    public void saveUser(User user) {
        user.setId(counter.incrementAndGet());
        users.add(user);
    }

    public void updateUser(User user) {
        int index = users.indexOf(user);
        users.set(index, user);
    }

    public void deleteUserById(long id) {

        for (Iterator<User> iterator = users.iterator(); iterator.hasNext(); ) {
            User user = iterator.next();
            if (user.getId() == id) {
                iterator.remove();
            }
        }
    }
```

```
    public boolean isUserExist(User user) {
        return findByName(user.getName())!=null;
    }

    public void deleteAllUsers(){
        users.clear();
    }

    private static List<User> populateDummyUsers(){
        List<User> users = new ArrayList<User>();
        users.add(new User(counter.incrementAndGet(),"Sam",30, 70000));
        users.add(new User(counter.incrementAndGet(),"Tom",40, 50000));
        users.add(new User(counter.incrementAndGet(),"Jerome",45, 30000));
        users.add(new User(counter.incrementAndGet(),"Silvia",50, 40000));
        return users;
    }

}
```

Project Structure

Finally, shown below is the project structure for this example.



Download Source Code

Download Now!