WebSystique
learn together

Secure Spring REST API using Basic Authentication

Created on: July 30, 2016 | **Last updated on:** September 30, 2017
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. Let's get going.

Other interesting posts you may like

- Secure Spring REST API using OAuth2
- Spring Boot+AngularJS+Spring Data+Hibernate+MySQL CRUD App
- Spring Boot REST API Tutorial
- Spring Boot WAR deployment example
- Spring Boot Introduction + Hello World Example

Recent Posts

Spring Boot + AngularJS + Spring Data + JPA CRUD App Example

Spring Boot Rest API Example

Spring Boot WAR deployment example

Spring Boot Introduction + hello world example

Secure Spring REST API using OAuth2

- Spring 4 Caching Annotations Tutorial
- Spring 4 Cache Tutorial with EhCache
- Spring 4 Email With Attachment Tutorial
- Spring 4 Email Integration Tutorial
- Spring MVC 4+JMS+ActiveMQ Integration Example
- Spring 4+JMS+ActiveMQ @JmsLister @EnableJms Example
- Spring 4+JMS+ActiveMQ Integration Example
- Spring MVC 4+Apache Tiles 3 Integration Example
- Spring MVC 4+Spring Security 4 + Hibernate Integration Example
- Spring MVC 4+AngularJS Example
- Spring MVC 4+AngularJS Routing with ngRoute Example
- Spring MVC 4+AngularJS Routing with UI-Router Example
- Spring MVC 4+Hibernate 4 Many-to-many JSP Example
- Spring MVC 4+Hibernate 4+MySQL+Maven integration example using annotations
- Spring MVC4 FileUpload-Download Hibernate+MySQL Example
- TestNG Mockito Integration Example Stubbing Void Methods
- Maven surefire plugin and TestNG Example
- Spring MVC 4 Form Validation and Resource Handling

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.

Basi
secu
with
requ
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.encodeBa

HttpHeaders headers = getHeaders();
headers.add("Authorization", "Basic " + base64ClientCredent
```

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.Autowir
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.authe
import org.springframework.security.config.annotation.web.b
import org.springframework.security.config.annotation.web.b
import org.springframework.security.config.annotation.web.c
import org.springframework.security.config.annotation.web.c
import org.springframework.security.config.http.SessionCrea

@Co
@En
pub
```



```
    @Autowired
    public void configureGlobalSecurity(AuthenticationManag
```

```
        auth.inMemoryAuthentication().withUser("bill").pass
        auth.inMemoryAuthentication().withUser("tom").passw
    }

    @Override
    protected void configure(HttpSecurity http) throws Exce

      http.csrf().disable()
        .authorizeRequests()
        .antMatchers("/user/**").hasRole("ADMIN")
        .and().httpBasic().realmName(REALM).authenticationE
        .and().sessionManagement().sessionCreationPolicy(Se
    }

    @Bean
    public CustomBasicAuthenticationEntryPoint getBasicAuth
        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.AuthenticationExce
import org.springframework.security.web.authentication.www.

public class CustomBasicAuthenticationEntryPoint extends Ba

    @Override
    public void commence(final HttpServletRequest request,
            final HttpServletResponse response,
            final AuthenticationException authException) th
        //Authentication failed, send error response.
        response.setStatus(HttpServletResponse.SC_UNAUTHORI
        response.addHeader("WWW-Authenticate", "Basic realm

        PrintWriter writer = response.getWriter();
        writer.println("HTTP Status 401 : " + authException
    }

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

That actio

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.Autowir
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.RequestMappi
import org.springframework.web.bind.annotation.RequestMetho
import org.springframework.web.bind.annotation.RestControll
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 d


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

    @RequestMapping(value = "/user/", method = RequestMetho
    public ResponseEntity<List<User>> listAllUsers() {
        List<User> users = userService.findAllUsers();
        if(users.isEmpty()){
            return new ResponseEntity<List<User>>(HttpStatu
        }
        return new ResponseEntity<List<User>>(users, HttpSt
    }


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

    @RequestMapping(value = "/user/{id}", method = RequestM
    public ResponseEntity<User> getUser(@PathVariable("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
            return new ResponseEntity<User>(HttpStatus.NOT_
        }
        return new ResponseEntity<User>(user, HttpStatus.OK
    }



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

    @RequestMapping(value = "/user/", method = RequestMetho
    public ResponseEntity<Void> createUser(@RequestBody Use
```



```
        userService.saveUser(user);
```

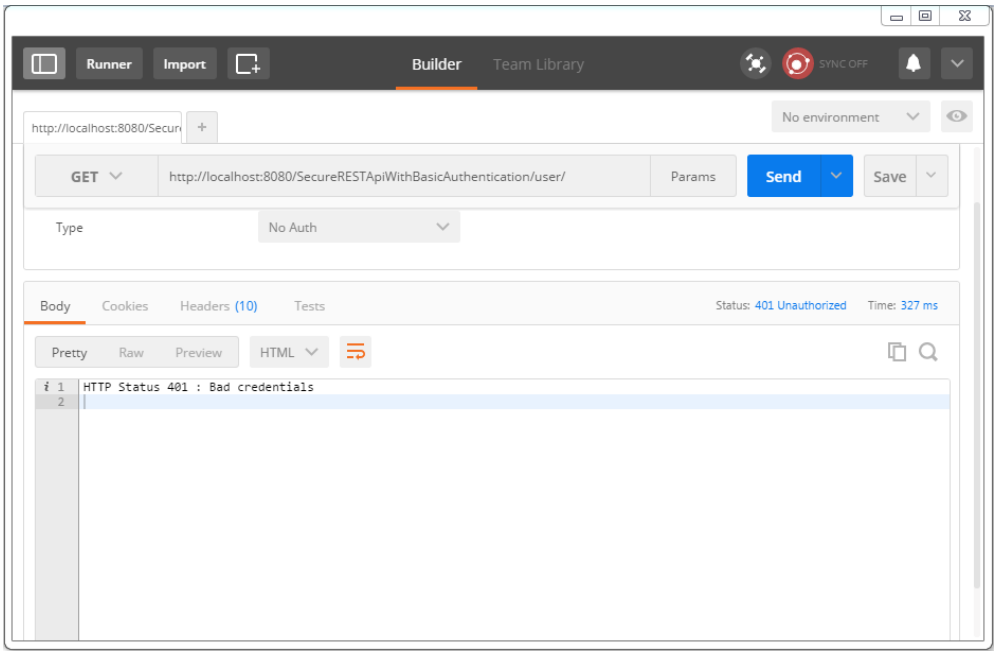
```
        HttpHeaders headers = new HttpHeaders();
        headers.setLocation(ucBuilder.path("/user/{id}").bu
        return new ResponseEntity<Void>(headers, HttpStatus
    }


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

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

        User currentUser = userService.findById(id);

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

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

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

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

    @RequestMapping(value = "/user/{id}", method = RequestM
    public ResponseEntity<User> deleteUser(@PathVariable("i
        System.out.println("Fetching & Deleting User with i

        User user = userService.findById(id);
        if (user == null) {
            System.out.println("Unable to delete. User with
            return new ResponseEntity<User>(HttpStatus.NOT_
        }

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


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

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

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

}
```

Running the application

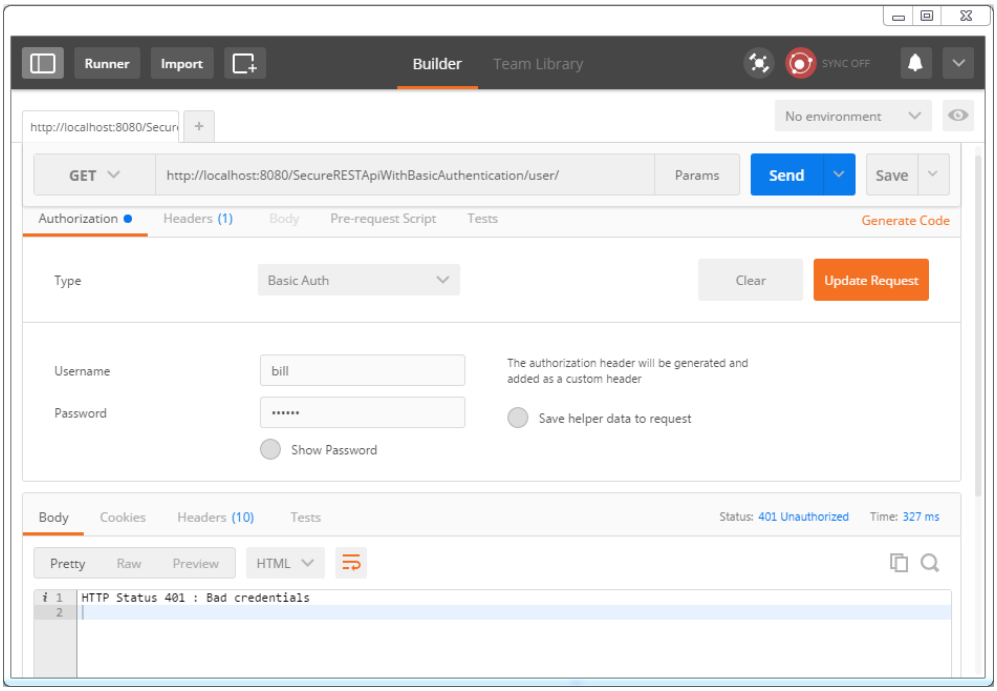
Build and deploy the application [on tomcat e.g]. Run it and test it using two differ

Usi

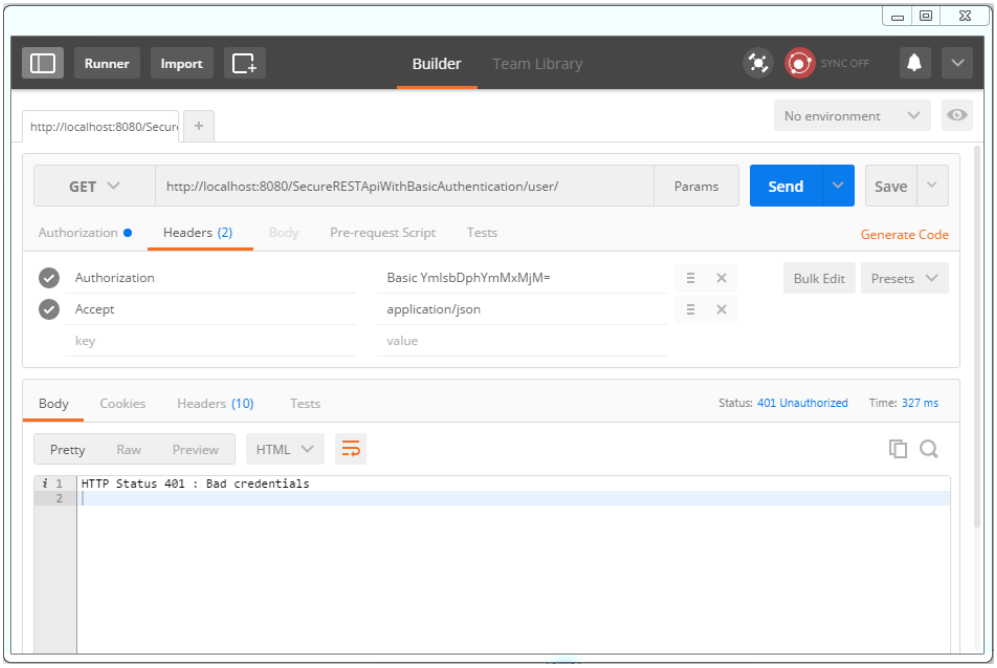
Send



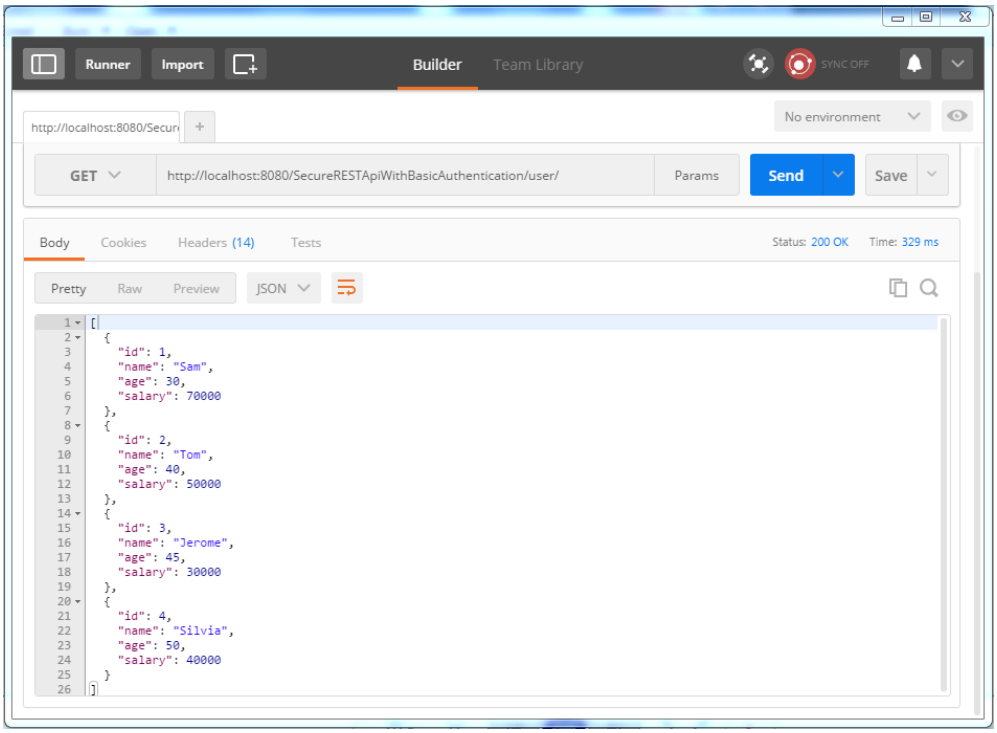
Now select type as 'Basic Auth' from dropdown, fill in username/password [bill/abc123], click on 'update request'.



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.

```
pacl

impo
impo
impo
impo
```



```
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://l

    /*
     * Add HTTP Authorization header, using Basic-Authentic
     */
    private static HttpHeaders getHeaders(){
        String plainCredentials="bill:abc123";
        String base64Credentials = new String(Base64.encode

        HttpHeaders headers = new HttpHeaders();
        headers.add("Authorization", "Basic " + base64Crede
        headers.setAccept(Arrays.asList(MediaType.APPLICATI
        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>
        ResponseEntity<List> response = restTemplate.exchan
        List<LinkedHashMap<String, Object>> usersMap = (Lis

        if(usersMap!=null){
            for(LinkedHashMap<String, Object> map : usersMa
                System.out.println("User : id="+map.get("id
            }
        }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>
        ResponseEntity<User> response = restTemplate.exchan
        User user = response.getBody();
        System.out.println(user);
    }

    /*
     * Send a POST request to create a new user.
     */
```



```
    }
```

```
    /*
     * 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>
        ResponseEntity<User> response = restTemplate.exchan
        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>
        restTemplate.exchange(REST_SERVICE_URI+"/user/3", H
    }


    /*
     * 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>
        restTemplate.exchange(REST_SERVICE_URI+"/user/", Ht
    }


    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
Use

Tes
Use

Tes
Loc

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 AtomicLon

    private static List<User> users;

    static{
        users= populateDummyUsers();
    }

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

    public User findById(long id) {
        for(User user : users){
```

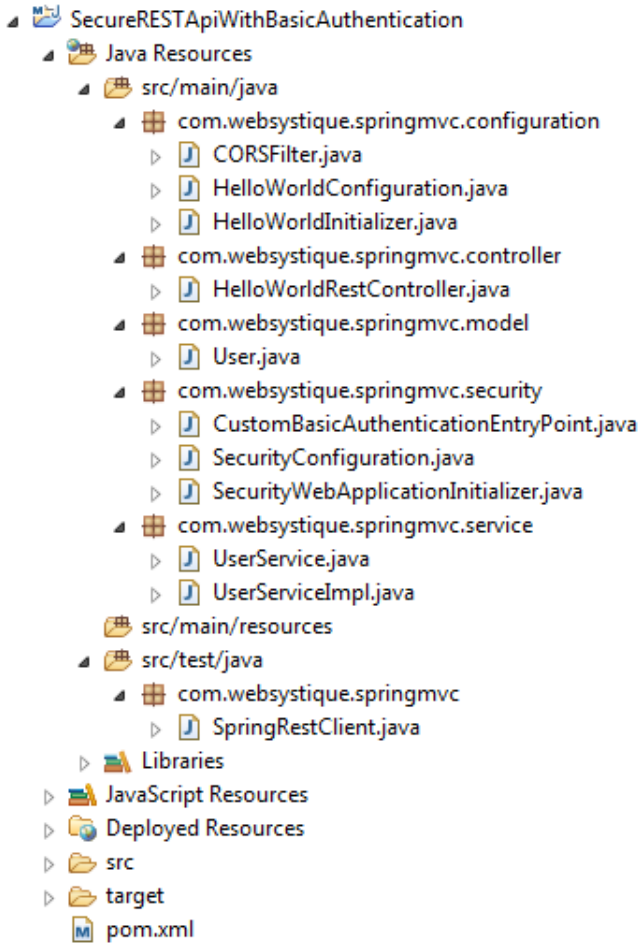


```
    public User findByName(String name) {
        for(User user : users){
```

```
            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(); it
            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",
        users.add(new User(counter.incrementAndGet(),"Tom",
        users.add(new User(counter.incrementAndGet(),"Jerom
        users.add(new User(counter.incrementAndGet(),"Silvi
        return users;
    }

}
```

Project Structure

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



Download Source Code

Download Now!

References

- Basic Authentication
- Spring Security 4 Project Page
- Spring Security 4 Reference Manual

websystiqueadmin

If you like tutorials on this site, why not take a step further and connect me on Facebook , Google Plus & Twitter as well? I would love to hear your thoughts on these articles, it will help improve further our learning process.

Rel

3. Spring Security 4 Secure View Fragments using tagliбs