[](https://www.devglan.com/)

* [Programming](https://www.devglan.com/programming/programming-tutorials)
* [Testing](https://www.devglan.com/testing/testing-tutorials)
* [AI](https://www.devglan.com/ai/artificial-intelligence-tutorials)
* [Devops](https://www.devglan.com/devops/devops-tutorials)
* [Data Science](https://www.devglan.com/datascience/datascience-tutorials)
* [Design](https://www.devglan.com/design/design-tutorials)
* [Blog](https://www.devglan.com/blogs)
* [Crypto Tools](https://www.devglan.com/cryptotools/cryptography-tools)
* [Dev Feed](https://www.devglan.com/)
* [Login](https://www.devglan.com/auth/login)

Top of Form

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Bottom of Form

[Story](https://www.devglan.com/me/add-story)

Spring Boot Security OAuth2 Example(Bcrypt Encoder)

   By [Dhiraj](https://www.linkedin.com/in/dhiraj-ray-devglan" \t "_blank),   Last updated on: 18 January, 2018203K

In this post we will be discussing about securing REST APIs using Spring Boot Security OAuth2 with an example.We will be implementing AuthorizationServer, ResourceServer and some REST API for different crud operations and test these APIs using Postman. For an integration with Angular, you can visit [Spring Boot OAuth2 Angular](https://www.devglan.com/spring-security/spring-boot-oauth2-angular).Here we will be using mysql database to read user credentials instead of in-memory authentication. Also, to ease our ORM solution, we will be using spring-data and BCryptPasswordEncoder for password encoding.Also, at the end we will make this configuration compatible with spring boot 2.

In this article, the authorization server and resource server is implemented using spring boot. For a 3rd party authorization server, you can visit this - [Spring Boot OAuth2 with Google](https://www.devglan.com/spring-security/spring-boot-security-google-oauth)

What is OAuth

OAuth is simply a secure authorization protocol that deals with the authorization of third party application to access the user data without exposing their password. eg. (Login with fb, gPlus, twitter in many websites..) all work under this protocol.The Protocol becomes easier when you know the involved parties. Basically there are three parties involved: oAuth Provider, oAuth Client and Owner.Here, oAuth Provider provides the auth token such as Facebook, twitter. Similarly, oAuth Client are the the applications which want access of the credentials on behalf of owner and owner is the user which has account on oAuth providers such as facebook and twitter.Here is an another article of [Securing REST API with Spring Boot Security Oauth2 JWT Token.](http://www.devglan.com/spring-security/spring-boot-oauth2-jwt-example)

What is OAuth2

[OAuth 2](https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2) is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

OAuth2 Roles

OAuth2 provides 4 different roles.

**Resource Owner**: User

**Client:** Application

**Resource Server**: API

**Authorization Server**: API

OAuth2 Grant Types

Following are the 4 different grant types defined by OAuth2

**Authorization Code:** used with server-side Applications

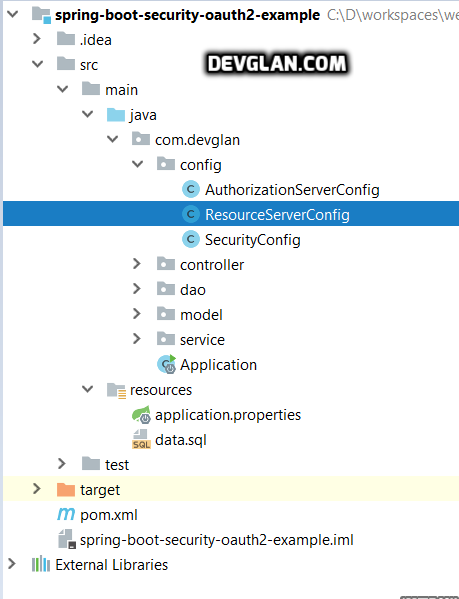
**Implicit:** used with Mobile Apps or Web Applications (applications that run on the user's device)

**Resource Owner Password Credentials:** used with trusted Applications, such as those owned by the service itself

**Client Credentials:** used with Applications API access

Project Structure

Following is the project structure of our Spring Boot Security OAuth2 implementation.

[](https://imgur.com/YgMjZEH)

Maven Dependencies

**pom.xml**

Following are the required dependencies.

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.8.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-security-oauth2</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

</dependency>

</dependencies>

OAuth2 Authorization Server Config

This class extends AuthorizationServerConfigurerAdapter and is responsible for generating tokens specific to a client.Suppose, if a user wants to login to [devglan.com](http://devglan.com/) via facebook then facebook auth server will be generating tokens for Devglan.In this case, Devglan becomes the client which will be requesting for authorization code on behalf of user from facebook - the authorization server.Following is a similar implementation that facebook will be using.

Here, we are using in-memory credentials with client\_id as devglan-client and CLIENT\_SECRET as devglan-secret.But you are free to use JDBC implementation too.

@EnableAuthorizationServer: Enables an authorization server.AuthorizationServerEndpointsConfigurer defines the authorization and token endpoints and the token services.

For an integration with Google along with a custom login, you can visit this article - [Spring Security OAuth2 Google Registration](https://www.devglan.com/spring-security/spring-security-oauth2-user-registration)

**AuthorizationServerConfig.java**

package com.devglan.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;

import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;

import org.springframework.security.oauth2.provider.token.TokenStore;

@Configuration

@EnableAuthorizationServer

public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

static final String CLIEN\_ID = "devglan-client";

static final String CLIENT\_SECRET = "devglan-secret";

static final String GRANT\_TYPE = "password";

static final String AUTHORIZATION\_CODE = "authorization\_code";

static final String REFRESH\_TOKEN = "refresh\_token";

static final String IMPLICIT = "implicit";

static final String SCOPE\_READ = "read";

static final String SCOPE\_WRITE = "write";

static final String TRUST = "trust";

static final int ACCESS\_TOKEN\_VALIDITY\_SECONDS = 1\*60\*60;

static final int FREFRESH\_TOKEN\_VALIDITY\_SECONDS = 6\*60\*60;

@Autowired

private TokenStore tokenStore;

@Autowired

private AuthenticationManager authenticationManager;

@Override

public void configure(ClientDetailsServiceConfigurer configurer) throws Exception {

configurer

.inMemory()

.withClient(CLIEN\_ID)

.secret(CLIENT\_SECRET)

.authorizedGrantTypes(GRANT\_TYPE\_PASSWORD, AUTHORIZATION\_CODE, REFRESH\_TOKEN, IMPLICIT )

.scopes(SCOPE\_READ, SCOPE\_WRITE, TRUST)

.accessTokenValiditySeconds(ACCESS\_TOKEN\_VALIDITY\_SECONDS).

refreshTokenValiditySeconds(FREFRESH\_TOKEN\_VALIDITY\_SECONDS);

}

@Override

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

endpoints.tokenStore(tokenStore)

.authenticationManager(authenticationManager);

}

}

OAuth2 Resource Server Config

Resource in our context is the REST API which we have exposed for the crud operation.To access these resources, client must be authenticated.In real-time scenarios, whenever an user tries to access these resources, the user will be asked to provide his authenticity and once the user is authorized then he will be allowed to access these protected resources.

@EnableResourceServer: Enables a resource server

**ResourceServerConfig.java**

package com.devglan.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

import org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigurer;

import org.springframework.security.oauth2.provider.error.OAuth2AccessDeniedHandler;

@Configuration

@EnableResourceServer

public class ResourceServerConfig extends ResourceServerConfigurerAdapter {

private static final String RESOURCE\_ID = "resource\_id";

@Override

public void configure(ResourceServerSecurityConfigurer resources) {

resources.resourceId(RESOURCE\_ID).stateless(false);

}

@Override

public void configure(HttpSecurity http) throws Exception {

http.

anonymous().disable()

.authorizeRequests()

.antMatchers("/users/\*\*").authenticated()

.and().exceptionHandling().accessDeniedHandler(new OAuth2AccessDeniedHandler());

}

}

**Other Interesting Posts**

[AES Encryption and Decryption in Java](https://www.devglan.com/corejava/java-aes-encypt-decrypt)

[Spring Boot Security JWT Authentication](https://www.devglan.com/spring-security/spring-boot-jwt-auth)

[Spring Boot Security REST Basic Authentication](https://www.devglan.com/spring-security/spring-boot-security-rest-basic-authentication)

[Spring Boot Actuator Complete Guide](https://www.devglan.com/spring-boot/spring-boot-actuator-tutorial-guide)

[Spring Boot Actuator Rest Endpoints Example](https://www.devglan.com/spring-boot/spring-boot-actuator-rest-endpoints-example)

[Spring Boot Security Hibernate Example with complete JavaConfig](https://www.devglan.com/spring-security/spring-boot-security-hibernate-login-example)

[Securing REST API with Spring Boot Security Basic Authentication](https://www.devglan.com/spring-security/spring-boot-security-rest-basic-authentication)

[Spring Boot Security Password Encoding using Bcrypt Encoder](https://www.devglan.com/spring-security/spring-boot-security-password-encoding-bcrypt-encoder)

[Websocket spring Boot Integration Without STOMP with complete JavaConfig](https://www.devglan.com/spring-boot/spring-websocket-integration-example-without-stomp)

Security Config

This class extends WebSecurityConfigurerAdapter and provides usual spring security configuration.Here, we are using bcrypt encoder to encode our passwords. You can try this [online Bcrypt Tool](http://www.devglan.com/online-tools/bcrypt-hash-generator) to encode and match bcrypt passwords.Following configuration basically bootstraps the authorization server and resource server.

@EnableWebSecurity : Enables spring security web security support.

@EnableGlobalMethodSecurity: Support to have method level access control such as @PreAuthorize @PostAuthorize

Here, we are using inmemorytokenstore but you are free to use JdbcTokenStore or JwtTokenStore.Here, is my another article to use [jwttokenstore with spring security OAUTH2](https://www.devglan.com/spring-security/spring-boot-oauth2-jwt-example)

**SecurityConfig.java**

package com.devglan.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.web.servlet.FilterRegistrationBean;

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.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.oauth2.provider.token.TokenStore;

import org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;

import org.springframework.web.cors.CorsConfiguration;

import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import org.springframework.web.filter.CorsFilter;

import javax.annotation.Resource;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Resource(name = "userService")

private UserDetailsService userDetailsService;

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Autowired

public void globalUserDetails(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService)

.passwordEncoder(encoder());

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable()

.anonymous().disable()

.authorizeRequests()

.antMatchers("/api-docs/\*\*").permitAll();

}

@Bean

public TokenStore tokenStore() {

return new InMemoryTokenStore();

}

@Bean

public BCryptPasswordEncoder encoder(){

return new BCryptPasswordEncoder();

}

@Bean

public FilterRegistrationBean corsFilter() {

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

CorsConfiguration config = new CorsConfiguration();

config.setAllowCredentials(true);

config.addAllowedOrigin("\*");

config.addAllowedHeader("\*");

config.addAllowedMethod("\*");

source.registerCorsConfiguration("/\*\*", config);

FilterRegistrationBean bean = new FilterRegistrationBean(new CorsFilter(source));

bean.setOrder(0);

return bean;

}

}

Rest APIs

Following are the very basic REST APIs that we have exposed for testing purpose.

**UserController.java**

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@RequestMapping(value="/user", method = RequestMethod.GET)

public List listUser(){

return userService.findAll();

}

@RequestMapping(value = "/user", method = RequestMethod.POST)

public User create(@RequestBody User user){

return userService.save(user);

}

@RequestMapping(value = "/user/{id}", method = RequestMethod.DELETE)

public String delete(@PathVariable(value = "id") Long id){

userService.delete(id);

return "success";

}

}

Now let us define the Userservice that is responsible for fetching user details from the database.Following is the implementation that spring will be using to validate user.

**UserServiceImpl.java**

@Service(value = "userService")

public class UserServiceImpl implements UserDetailsService, UserService {

@Autowired

private UserDao userDao;

public UserDetails loadUserByUsername(String userId) throws UsernameNotFoundException {

User user = userDao.findByUsername(userId);

if(user == null){

throw new UsernameNotFoundException("Invalid username or password.");

}

return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), getAuthority());

}

private List getAuthority() {

return Arrays.asList(new SimpleGrantedAuthority("ROLE\_ADMIN"));

}

public List findAll() {

List list = new ArrayList<>();

userDao.findAll().iterator().forEachRemaining(list::add);

return list;

}

}

Default Scripts

Following are the insert statements that are inserted when application starts.

INSERT INTO User (id, username, password, salary, age) VALUES (1, 'Alex123', '$2a$04$I9Q2sDc4QGGg5WNTLmsz0.fvGv3OjoZyj81PrSFyGOqMphqfS2qKu', 3456, 33);

INSERT INTO User (id, username, password, salary, age) VALUES (2, 'Tom234', '$2a$04$PCIX2hYrve38M7eOcqAbCO9UqjYg7gfFNpKsinAxh99nms9e.8HwK', 7823, 23);

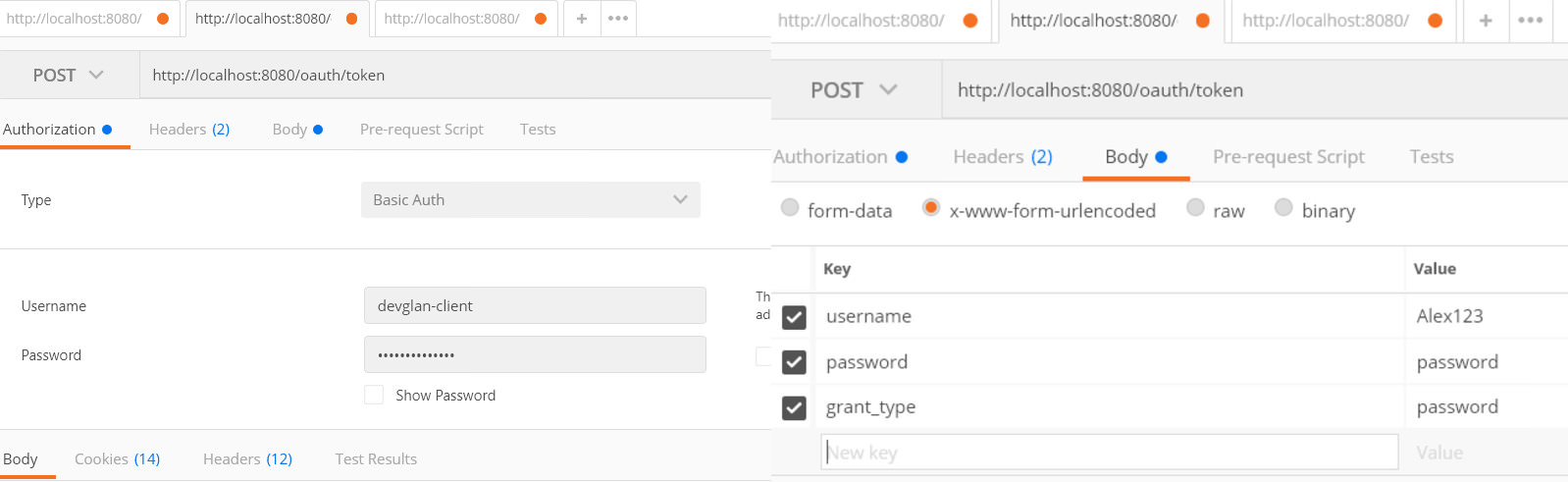
INSERT INTO User (id, username, password, salary, age) VALUES (3, 'Adam', '$2a$04$I9Q2sDc4QGGg5WNTLmsz0.fvGv3OjoZyj81PrSFyGOqMphqfS2qKu', 4234, 45);

Testing the Application

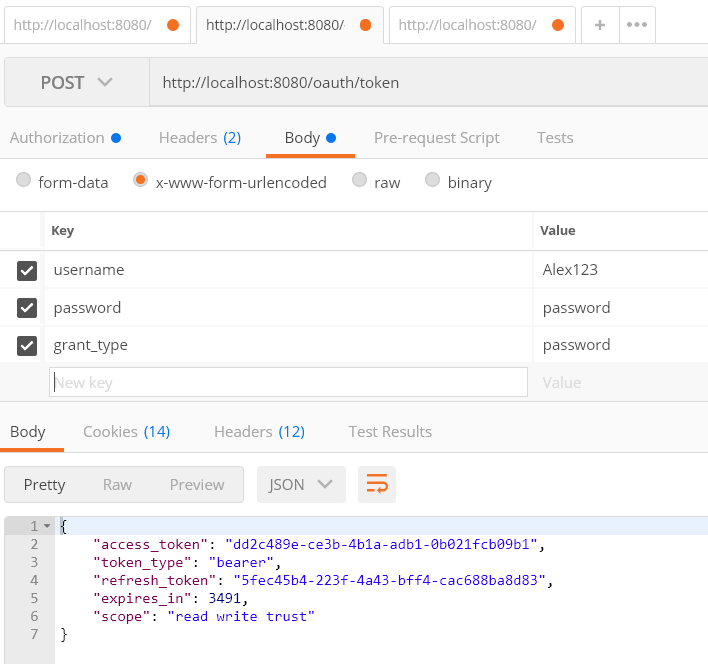
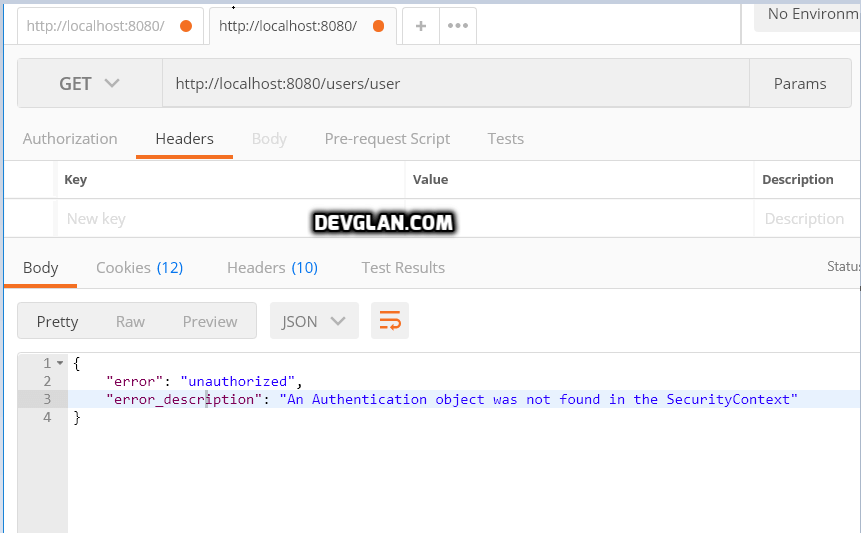
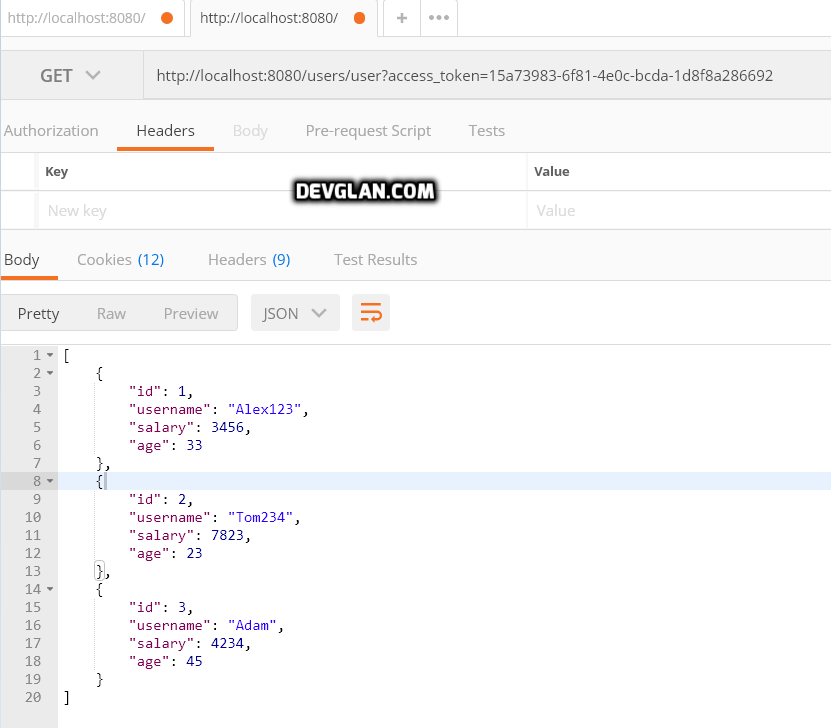
Here, we will test the app with Postman. If you wish to see the angular client, you can visit my another article here - [Spring Boot OAUTH2 Angular Example](https://www.devglan.com/spring-security/spring-boot-oauth2-angular)

Run Application.java as a java application.We will be using postman to test the OAuth2 implementation.

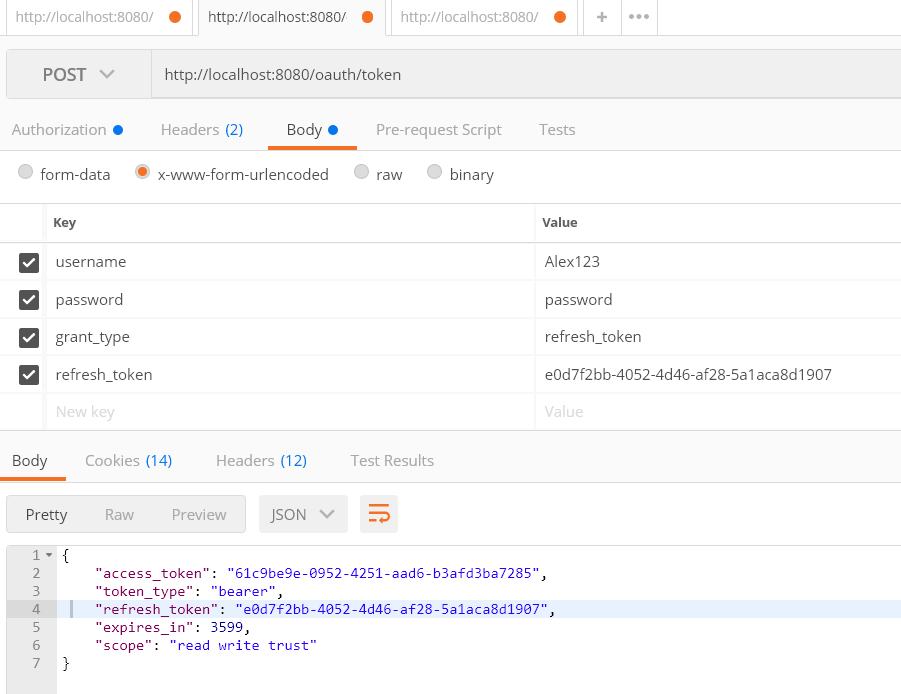
**Generate AuthToken**:In the header we have username and password as Alex123 and password respectively as Authorization header.As per Oauth2 specification, Access token request should use application/x-www-form-urlencoded.Following is the setup.

[](https://imgur.com/1T0dhV7)

Once you make the request you will get following result.It has access token as well as refresh token.

[](https://imgur.com/sP3uLm8)**Accessing Resource Without Token** [](https://imgur.com/FgfpYhD)**Accessing Resource With Token** [](https://imgur.com/g6WphWD)**Using refresh token to refresh the token**

Usually, the token expiry time is very less in case of oAuth2 and you can use following API to refresh token once it is expired.

[](https://imgur.com/ugZMxX7)

Common Errors

I can see in the comment section that there are 2 errors which most of the readers have encountered.Hence,adding this section that ideally should help readers.

**Full authentication is required to access this resource**

{

"timestamp": 1513747665246,

"status": 401,

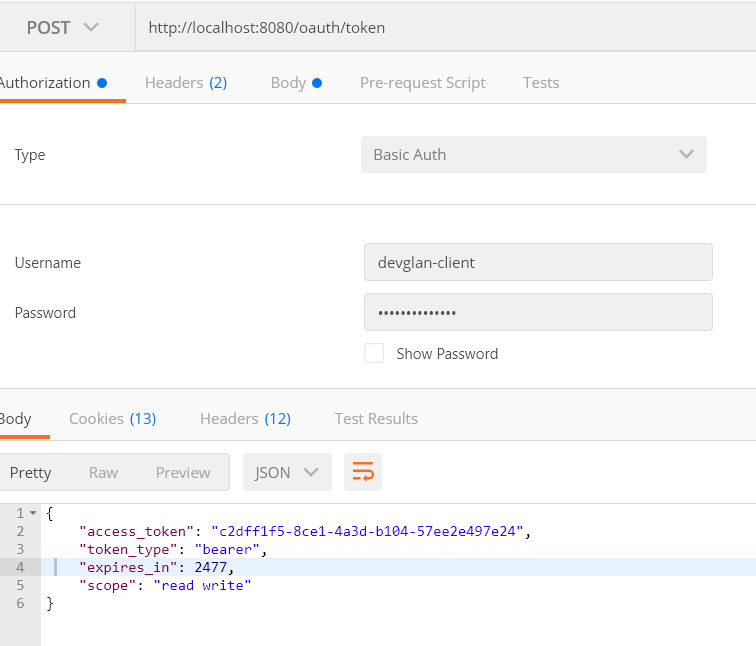
"error": "Unauthorized",

"message": "Full authentication is required to access this resource",

"path": "/oauth/token"

}

It causes in case you have missed to add username/password under authorization section of postman.Under this section, select Type as Basic Auth and provide the credentials as devglan-client and devglan-secret and then make the request to url - [http://localhost:8080/oauth/token to get the auth token.Following is the screenshot.](https://www.devglan.com/)

[](https://imgur.com/a08sRAx)**There is no client authentication. Try adding an appropriate authentication filter.**

{

"error": "unauthorized",

"error\_description": "There is no client authentication. Try adding an appropriate authentication filter."

}

In this case check your auth url.It should be - [http://localhost:8080/oauth/token](https://www.devglan.com/) instead of [http://localhost:8080/oauth/token/](https://www.devglan.com/)

**Missing grant type.**

{

"error": "invalid\_request",

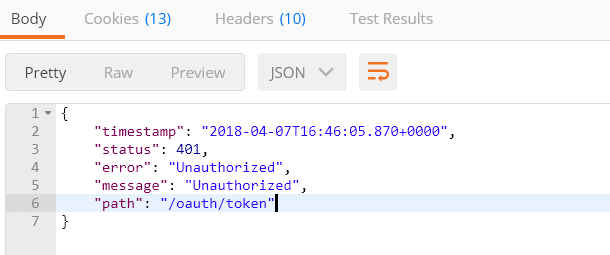
"error\_description": "Missing grant type"

}

In this case you have missed adding grant\_type in the request.Try adding it as **password**

Spring Boot 2 OAUTH2

While running this application with above configurations in Spring Boot 2, you will find below error.

[](https://imgur.com/1BzH6JX)

Following are the changes in pom.xml to make this example work with spring boot 2.

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

</parent>

...

<dependency>

<groupId>org.springframework.security.oauth</groupId>

<artifactId>spring-security-oauth2</artifactId>

<version>2.0.10.RELEASE</version>

</dependency>

For Spring Boot 2 you need to Bcrypt CLIENT\_SECRET,so in AuthorizationServerConfig.java change line 17 into:

static final String CLIENT\_SECRET = "$2a$04$e/c1/RfsWuThaWFCrcCuJeoyvwCV0URN/6Pn9ZFlrtIWaU/vj/BfG";

Conclusion

In this tutorial we learned about securing REST API with OAUTH2 with implementation of resouce server and authorisation server. If you have anything that you want to add or share then please share it below in the **comment section**.You can download the source from [github.](https://github.com/only2dhir/spring-boot-security-oauth2/tree/master)

If You Appreciate This, You Can Consider:

Top of Form

 https://www.paypalobjects.com/en_US/i/scr/pixel.gif

Bottom of Form

* Like us at: [[Facebook](https://goo.gl/YtUNxk)](https://goo.gl/YtUNxk)or follow us at [Twitter](https://goo.gl/sLbAWK)
* Share this article on social media or with your teammates.

About The Author



A technology savvy professional with an exceptional capacity to analyze, solve problems and multi-task. Technical expertise in highly scalable distributed systems, self-healing systems, and service-oriented architecture. Technical Skills: Java/J2EE, Spring, Hibernate, Reactive Programming, Microservices, Hystrix, Rest APIs, Java 8, Kafka, Kibana, Elasticsearch, etc.

Further Reading on Spring Security

[1. Add Spring Security Custom Filter](https://www.devglan.com/spring-security/add-spring-security-custom-filter)

[2. Spring Boot Oauth2 Angular](https://www.devglan.com/spring-security/spring-boot-oauth2-angular)

[3. Jwt Role Based Authorization](https://www.devglan.com/spring-security/jwt-role-based-authorization)

[4. Angular Jwt Authentication](https://www.devglan.com/spring-security/angular-jwt-authentication)

[5. Spring Boot Jwt Auth](https://www.devglan.com/spring-security/spring-boot-jwt-auth)

[6. Spring Boot Security Rest Basic Authentication](https://www.devglan.com/spring-security/spring-boot-security-rest-basic-authentication)

[7. Spring Boot Security Custom Form Login Example](https://www.devglan.com/spring-security/spring-boot-security-custom-form-login-example)

[8. Spring Boot Security Hibernate Login Example](https://www.devglan.com/spring-security/spring-boot-security-hibernate-login-example)

[9. Spring Jms Activemq Integration Example](https://www.devglan.com/spring-mvc/spring-jms-activemq-integration-example)

[10. Spring Mvc Angularjs Integration Example](https://www.devglan.com/spring-mvc/spring-mvc-angularjs-integration-example)

Recommended

* [Add Spring Security Custom Filter](https://www.devglan.com/spring-security/add-spring-security-custom-filter)
* [Spring Boot Oauth2 Angular](https://www.devglan.com/spring-security/spring-boot-oauth2-angular)
* [Jwt Role Based Authorization](https://www.devglan.com/spring-security/jwt-role-based-authorization)
* [Angular Jwt Authentication](https://www.devglan.com/spring-security/angular-jwt-authentication)
* [Spring Boot Jwt Auth](https://www.devglan.com/spring-security/spring-boot-jwt-auth)
* [Spring Boot Security Rest Basic Authentication](https://www.devglan.com/spring-security/spring-boot-security-rest-basic-authentication)
* [Spring Boot Security Custom Form Login Example](https://www.devglan.com/spring-security/spring-boot-security-custom-form-login-example)
* [Spring Boot Security Hibernate Login Example](https://www.devglan.com/spring-security/spring-boot-security-hibernate-login-example)
* [Spring Jms Activemq Integration Example](https://www.devglan.com/spring-mvc/spring-jms-activemq-integration-example)
* [Spring Mvc Angularjs Integration Example](https://www.devglan.com/spring-mvc/spring-mvc-angularjs-integration-example)
* [[Facebook](https://www.facebook.com/sharer/sharer.php?u=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example)Facebook](https://www.facebook.com/sharer/sharer.php?u=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example)
* [[Twitter](https://twitter.com/intent/tweet?text=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example)Twitter](https://twitter.com/intent/tweet?text=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example)
* [[Pinterest](http://pinterest.com/pin/create/button/?url=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example&media=http://www.devglan.com/style/img/logo.png&description=Implementation%20of%20Spring%20Boot%20Security%20OAuth2%20with%20CRUD%20example%20to%20secure%20REST%20APIs.Implementation%20of%20AuthorizationServer,ResourceServer%20with%20mysql%20db%20and%20spring%20data.%20BCryptPasswordEncoder%20is%20used%20for%20password%20encoding.)Pinterest](http://pinterest.com/pin/create/button/?url=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example&media=http://www.devglan.com/style/img/logo.png&description=Implementation%20of%20Spring%20Boot%20Security%20OAuth2%20with%20CRUD%20example%20to%20secure%20REST%20APIs.Implementation%20of%20AuthorizationServer,ResourceServer%20with%20mysql%20db%20and%20spring%20data.%20BCryptPasswordEncoder%20is%20used%20for%20password%20encoding.)
* [[Linkedin](https://www.linkedin.com/sharing/share-offsite/?url=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example&title=Spring%20Boot%20Security%20OAuth2%20Example%20|%20DevGlan&summary=Implementation%20of%20Spring%20Boot%20Security%20OAuth2%20with%20CRUD%20example%20to%20secure%20REST%20APIs.Implementation%20of%20AuthorizationServer,ResourceServer%20with%20mysql%20db%20and%20spring%20data.%20BCryptPasswordEncoder%20is%20used%20for%20password%20encoding.)Linkedin](https://www.linkedin.com/sharing/share-offsite/?url=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example&title=Spring%20Boot%20Security%20OAuth2%20Example%20|%20DevGlan&summary=Implementation%20of%20Spring%20Boot%20Security%20OAuth2%20with%20CRUD%20example%20to%20secure%20REST%20APIs.Implementation%20of%20AuthorizationServer,ResourceServer%20with%20mysql%20db%20and%20spring%20data.%20BCryptPasswordEncoder%20is%20used%20for%20password%20encoding.)
* [[Tumblr](http://tumblr.com/share/link?url=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example)Tumblr](http://tumblr.com/share/link?url=https://www.devglan.com/spring-security/spring-boot-security-oauth2-example)



Devglan is one stop platform for all  
programming tutorials and courses.

About Us

* [About Us](https://www.devglan.com/about-us)
* [Contact Us](https://www.devglan.com/user/contact-us)
* [Submission Criteria](https://www.devglan.com/user/submission-criteria)
* [Privacy Policy](https://www.devglan.com/privacy)

Quick Links

* [Home](https://www.devglan.com/)
* [Login / Join](https://www.devglan.com/auth/login)
* [Submit Your Story](https://www.devglan.com/me/add-story)
* [Donate](https://www.devglan.com/donate-to-devglan)

Contact Us

**Dhiraj**  
[dhiraj@devglan.com](mailto:)

© 2020 Devglan. All rights reserved.

Left and Right View of Binary Tree in Java [Read Now!](https://www.devglan.com/datastructure/binary-tree-left-view-javahttps:/www.devglan.com/datastructure/binary-tree-left-view-java)