LAB- Using Custom Authorization Server

Step 1 – Creating a Custom Authorization Server

In this step you will be working on **04-authorization-server-start** project.

Open pom.xml and observe that we have added

```
<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-oauth2</artifactId>
      </dependency>
```

Authorization Server exposes **endpoints for**
requesting access token,
checking the access token,
Authorizing the client,etc

Press CTRL+SHIFT+T and open **EnableAuthorizationServer.java**

Observe the It is importing `AuthorizationServerEndpointsConfiguration`

Open it and observe that it is defining beans for authorizing the client, requesting access token, checking access token, etc

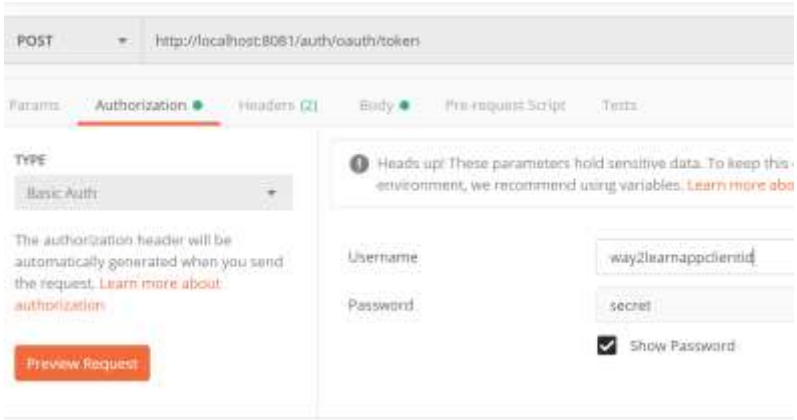
Also Observe `AuthorizationServerSecurityConfiguration.java.` This contains the default security configurations for OAuthServer.

Open AuthorizationServerConfiguration.java given to you in com.way2learnonline package and complete TODO-1 ,TODO-2 and TODO-3

Open application.yml and observe that we have configured context-path as /auth

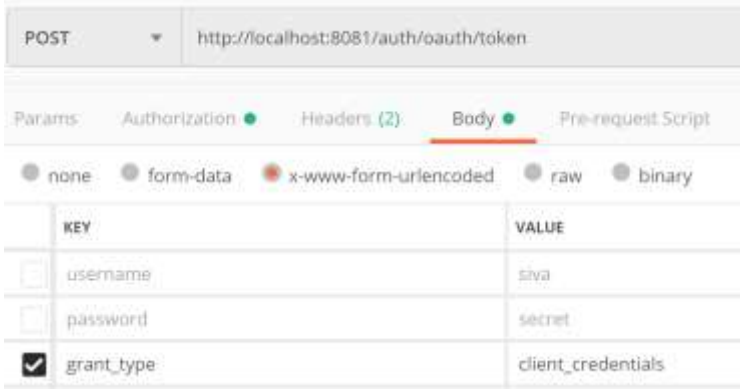
Start this application from boot dashboard and use postman to make a POST request to **http://localhost:8081/auth/oauth/token**

Select Authorization tab in postman, select Basic Auth in the drop down and give `way2learnappclientid/secret` as username/password and click Preview Request.



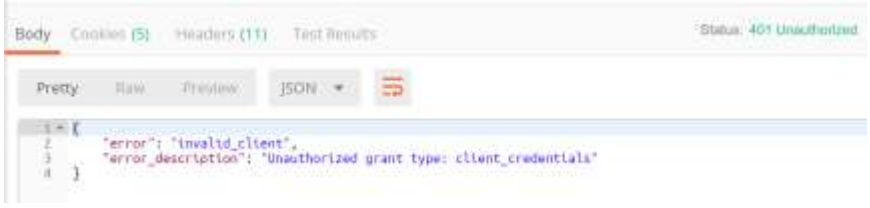
Observe that Authorization header will be added in post man.

Select Body and select x-www.form-url-encoded and give grant_type as client_credentials as shown below:



Now click on send in post man to make a request.

You should get a response as shown below:



Why?

Because for client with id `way2learnappclientid` we have not enabled client_credentials as authorized grant type.

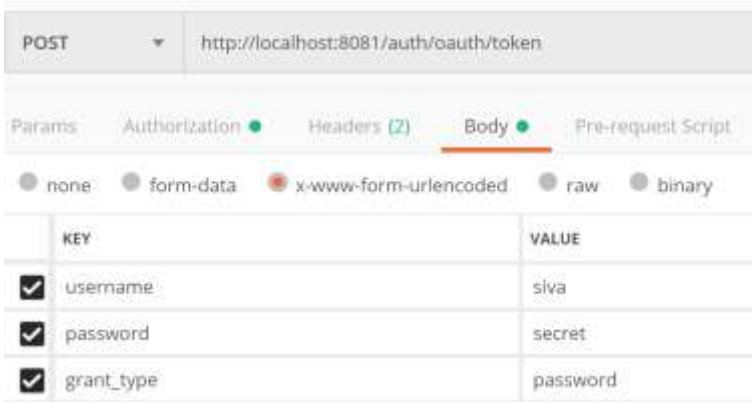
Try the same with client id microclient. You should get the access token.

Now we want the client to use “password” grant type.

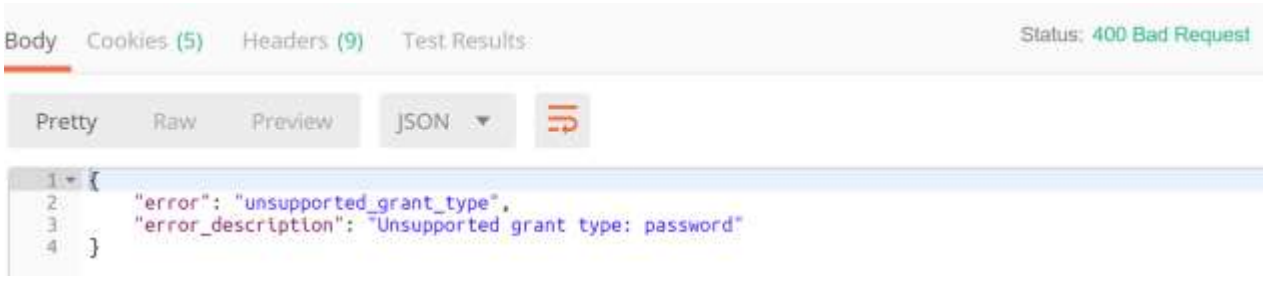
So, we need to configure user credentials just like how we do in Normal Spring security.

Open WebsecurityConfiguration.java in com.way2learnonline package and complete TODO-4
Observe that we have configured siva/secret as username/password

Restart the application and give a reques in postman but now, pass grant_type as password and also username, password as shown below : (Make sure that u still pass client_id and secreat as Authorization Header like earlier)



You should get an error as shown below :



Why?

We have to give reference of authentication Manager to AuthorizationServerEndpoints to that they can validate the user credentials using authentication manager

Now Open AuthorizationServerConfiguration.java in com.way2learnonline package and complete TODO-5

Now restart the application and try to get the token using password grant type.

You should be able to get the token using password grant.

Now we want to check if token is valid and get the details of token

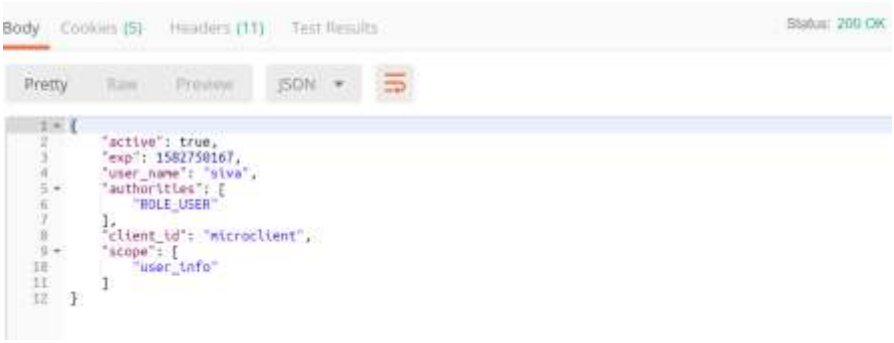
In postman, make a post request to http://localhost:8081/auth/oauth/check_token?token=<<token>>

You should get the below error because by default check_token access is denied for all.

Now Open AuthorizationServerConfiguration.java in com.way2learnonline package and complete TODO-6

In postman, make a post request to http://localhost:8081/auth/oauth/check_token?token=<<token>>

You should get the details of token like below:



Now configure checkTokenAccess to be granted only if client credentials are provided by configure as shown below:

```
@Override
public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {

    oauthServer
          .checkTokenAccess("isAuthenticated()")
         ;

}
```

Now we want to configure our authorization server to generate JWT tokens.

For generating JWT tokens, we need a key pair.

So, execute the following command to generate keystore file with a keypair.

keytool -genkey -alias tomcat -storetype jks -keyalg RSA -keysize 2048 -keystore mykeystore.jks

Give the password as “password”

Copy the generated mykeystore.p12 into src/main/resources

Now Open AuthorizationServerConfiguration.java in com.way2learnonline package and complete TODO-7 and TODO-8

Restart the application and make a request to get the token just like you have done in earlier steps.

You should get JWT token

Now make a request to check token. You should be able to see the token details.

Step 2 – Customizing Authorization Server to support authorization code grant

Now we want to configure Authorization server to support authorization code grant.

When ever client application wants the resource owner to authorize, client will redirect the browser to /oauth/authorize.

If authorization server needs to authorize, it should know who is the resource owner. So, /oauth/authorize endpoint of authorization server should be secured . So, a login page should be shown when the call comes to /oauth/authorize.

Open WebSecurityConfiguration.java in com.way2learnonline package and complete TODO-9

Once the Oauth client gets the token, it tries to get the user info by hitting userinfo endpoint.

We have already written an userinfo endpoint. Just open UserInfoController.java in

com.way2learnonline package and observe the code.

This /userinfo is a resource which should be accessible only if the request contains a token. So, We want some security filters to be configured to verify if the token is present in the request and if the token is valid. So, we need to enable resource server.

Open **MyResourceServerConfiguration.java** in com.way2learnonline package and complete TODO-9

Congratulations!!! Our Authorization Server is ready!!!

Now we need to configure client application to use out authorization server with authorization code grant flow

Now open application.yml of **03-security-google-oauth2-start** and configure as shown below:

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id:
694424912843-djndtscpp9mnomfhl3q8q9kk23jvilbf.apps.googleusercontent.com
            client-secret: Gj_w1C2xiXW-vC4WenJUvaB5
            scope: openid,profile,email
            client-name: SivaGoogle
          way2learnclient:
            client-id: way2learnappclientid
            client-secret: secret
            client-name: Way2learn
            scope: user_info
            redirect-uri:
https://localhost:8443/myapp/login/oauth2/code/way2learnappclient
            client-authentication-method: basic
            authorization-grant-type: authorization_code
            provider: way2learn-authserver-provider
        provider:
            way2learn-authserver-provider:
              token-uri: http://localhost:8081/auth/oauth/token
              authorization-uri: http://localhost:8081/auth/oauth/authorize
              user-info-uri: http://localhost:8081/auth/userinfo
              user-name-attribute: username
```

Now open login.html under src/main/resources/templates folder and complete TODO-9

You are done with client application Configuration also.

Now start **03-security-google-oauth2-start** application and give a request to http://localhost:8080/myapp/hello

You should be redirected to login page.

Now click on “Login with Way2learn Auth Server” link.

You should be redirected to the login page of Way2learn Auth Server.

Give the credentials as siva/secret. You should be logged in and should get redirected to

https://localhost:8443/myapp/hello

Congratulations!! You know how to configure client to use your custom authorization Server

Step 3 – Configuring a Resource Server

Now we want to to create a resource server which exposes rest api and secure it such that any request which is coming to this resource server should have a token.

Token has to be validated and if token has the required scopes, then only our rest controller should be accessible.

In this step, you will be working on **04-secured-micro-resource-server-start** project

We want to enable some filters on the resource server which will validate the incoming requests

@EnableResourceServer configures all the required filters for a resource server.

Open MyResourceServerConfiguration.java and complete TODO-1,TODO-2 ,TODO-3 and TODO-4

Open Application.java and complete TODO-5

Again open MyResourceServerConfiguration.java and complete TODO-6

Now start **04-secured-micro-resource-server-start** and observe that server starts on 8083

Open postman and give a request to http://localhost:8084/hello
You should not be granted access as we have secured in TODO-4 in MyResourceServerConfiguration.java

Give a request from postman to get token from authorization server as you have done in earlier exercise

Then give a request to http://localhost:8084/hello?access_token=<<token u got in earlier step>> You should be granted access.

We don't want to configure RemoteTokenServices bean as we have done in TODO-5 in Application.java. We want the same bean to be configured by writing in application.yml

Open Application.java and comment the bean definition for RemoteTokenServices

Configure the following in application.yml

```
security:
  oauth2:
    client:
      clientId: microclient
      clientSecret: secret
    resource:
      tokenInfoUri: http://localhost:8081/auth/oauth/check_token
      user-info-uri:  http://localhost:8081/auth/userinfo
```

Do you remember that you have configured check token access for anyone who is authenticated as shown below in Authorization Server application in :

```
@Override
    public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws
Exception {

             oauthServer//.tokenKeyAccess("permitAll")
              .checkTokenAccess("isAuthenticated()")
            ;

    }
```

Now restart **04-secured-micro-resource-server-start** and give a request to to http://localhost:8084/hello?access_token=<<token u got in earlier step>>

You should still be granted access.

In this case resource server is making a request to tokenInfoUri configured and also getting userdetails by hitting user-info-uri. Many network calls.

Actually, we have configured our authorization server to generate JWT token.

JWT token can be verified locally by ResourceServer if it has the public key of AuthorizarionServer

So, Configure application.yml as shown below to configure JWT key uri from where resource server can get public key from authorization server during startup

```
security:
  oauth2:
    client:
      clientId: microclient
      clientSecret: secret
    resource:
#       tokenInfoUri: http://localhost:8081/auth/oauth/check_token
#       user-info-uri:  http://localhost:8081/auth/userinfo
      jwt:
        key-uri: http://localhost:1234/auth/oauth/token_key
```

In 04-authorization-server-start, open AuthorizationServerConfiguration.java and configure tokenKeyAccess as shown below:

```
      @Override
    public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws
Exception {

             oauthServer.tokenKeyAccess("isAuthenticated()")
              .checkTokenAccess("isAuthenticated()")
            ;

    }
```

Now restart **04-secured-micro-resource-server-start** and give a request to to http://localhost:8084/hello?access_token=<<token u got in earlier step>>

You should still be granted access.

Congratulations !! You are ready to secure your apps using OAuth2 and Custom Authorization Server