**Term Project Part 3, Assignment3.1**

In the Assignment3.1, you’ll examine different aspects of Microservices Security.

Below is the application directory structure.

**A close up of a map

Description automatically generated**

Note: While working on this Assignment:

* do not use the same **organizations** table columns values as in the Chapter 7 of the text book. You can use those values from this document or any others.
* do not use the same users **john.carnell** and **william.wooodward** that were used inChapter 7 of the text book. You can use the same users as in this this document or any others.

To set up the **EagleEye** OAuth2 authentication service you’ll needtoadd **spring-cloud-security** and **spring-security-oauth2** to the **authentication-service/pom.xml** file.

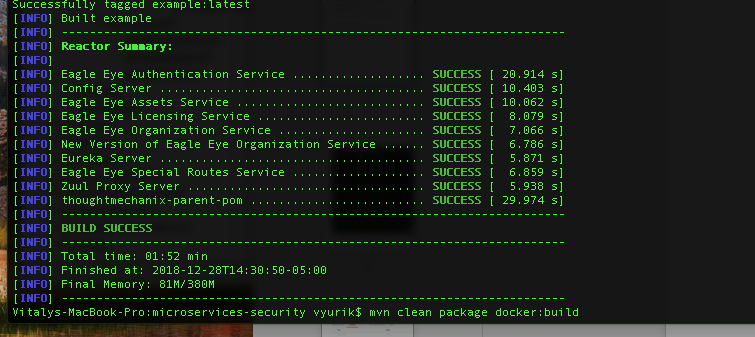
To register the EagleEye application with your authentication service, you’ll need to add class OAuth2Config.java to your authentication service.

Define and store application-level key name as **eagleeye** and secret as **thisissecret.**

**Clean the memory with command:**

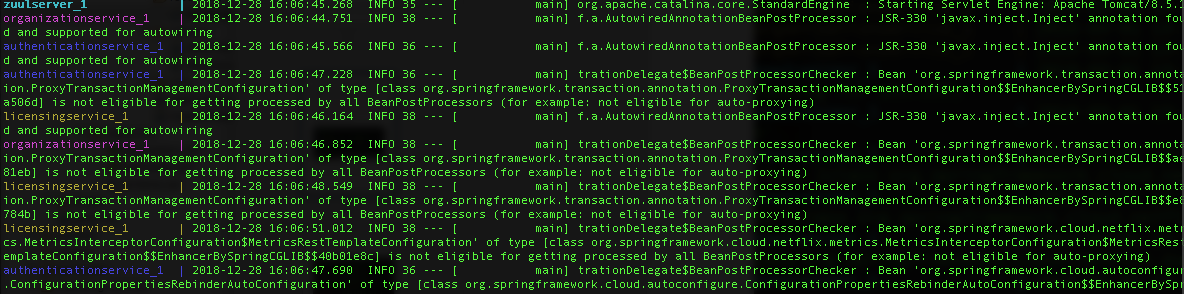
**docker system prune --all --force --volumes**

Build the application with command: **mvn clean package docker:build**

****

Run the application with command

**docker-compose -f docker/common/docker-compose.yml up**

****

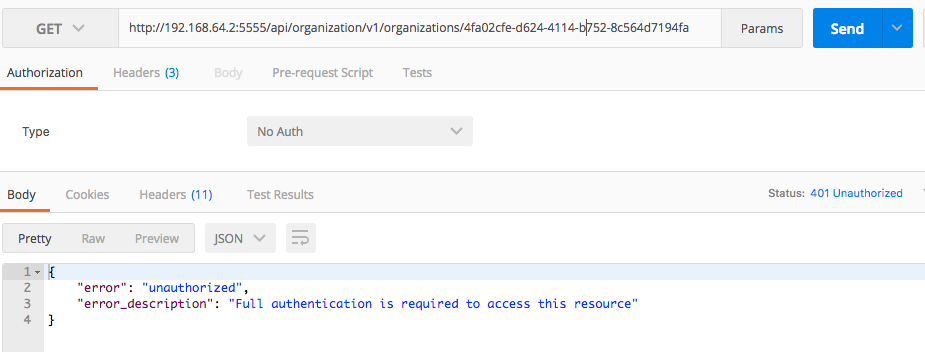
***Use Case 1: Restricting access to only authenticated users*  
In the first use case, we restrict access to authenticated users only and uncomment the first configure method of the organization service’s ResourceServerConfiguration class located at the following path:**

**organization - service/src/main/java/com/thoughtmec hanix/organization/security/ ResourceServerConfiguration.java**

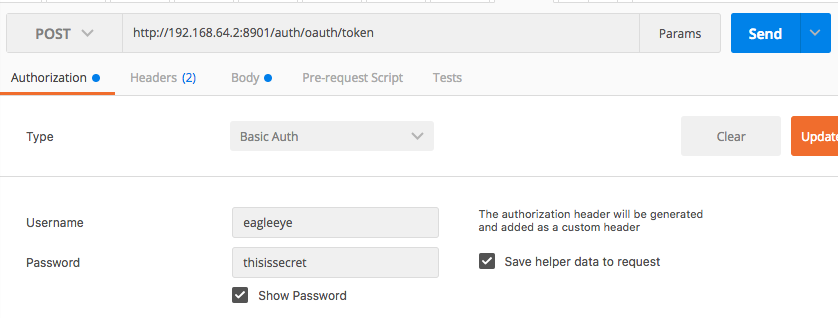
@Override **public void** configure(HttpSecurity http) **throws** Exception{ http.authorizeRequests().anyRequest().authenticated(); }

Upon issuing the following GET request without supplying authentication credentials http://<my\_docker\_ip>:5555/api/organization/v1/organizations/4fa02cfe - d624 - 4114 - b752 - 8c564d7194fa

we receive the following result:

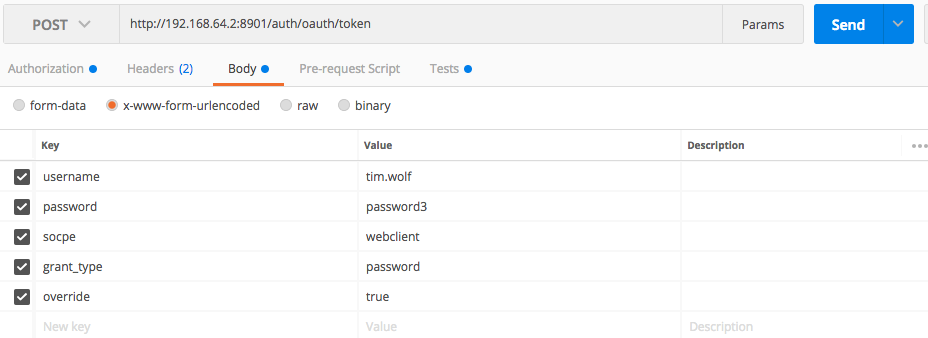


In the next section, we prepare to authenticate against the authenti cation server supplying the following Username: **eagleeye** and Password: **thisissecret** credentials **:**



Next, you define two users in the authentication - service’s WebSecurityConfigurer class’s configure method and make the approptriate changes to **authentication-service’s** **schema.sql.**

Now you are ready to create an OAuth 2 bearer token for the user **tim.wolf** with the USER role.

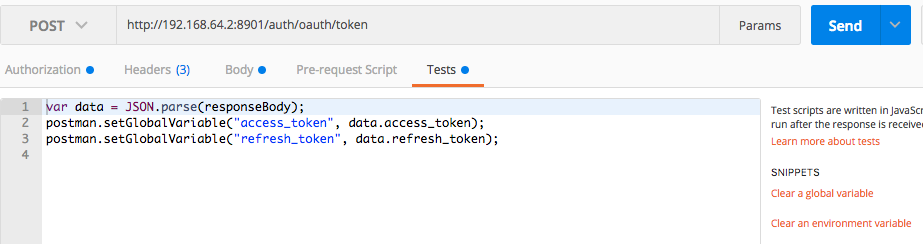


Add the following to the **Tests** section of the Postman:

var data = JSON.parse(responseBody);

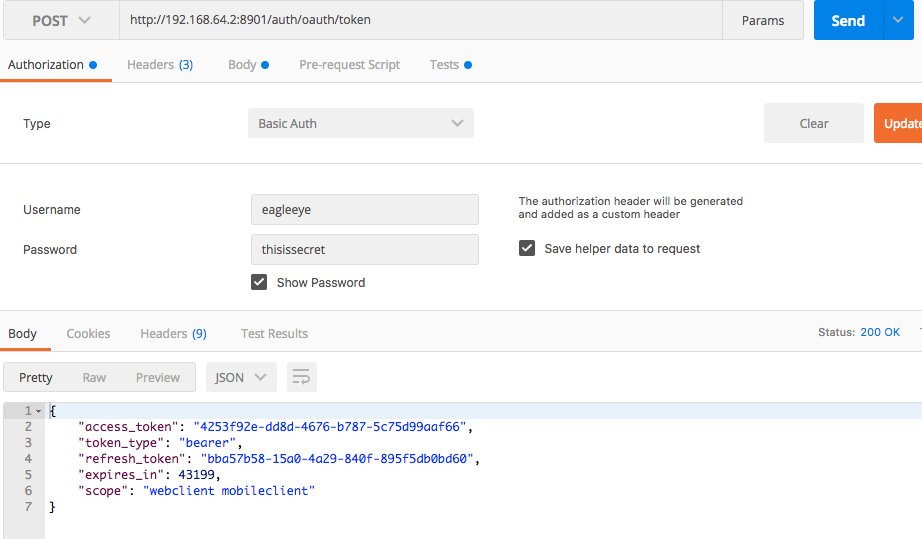
postman.setGlobalVariable("access\_token", data.access\_token);

postman.setGlobalVariable("refresh\_token", data.refresh\_token);



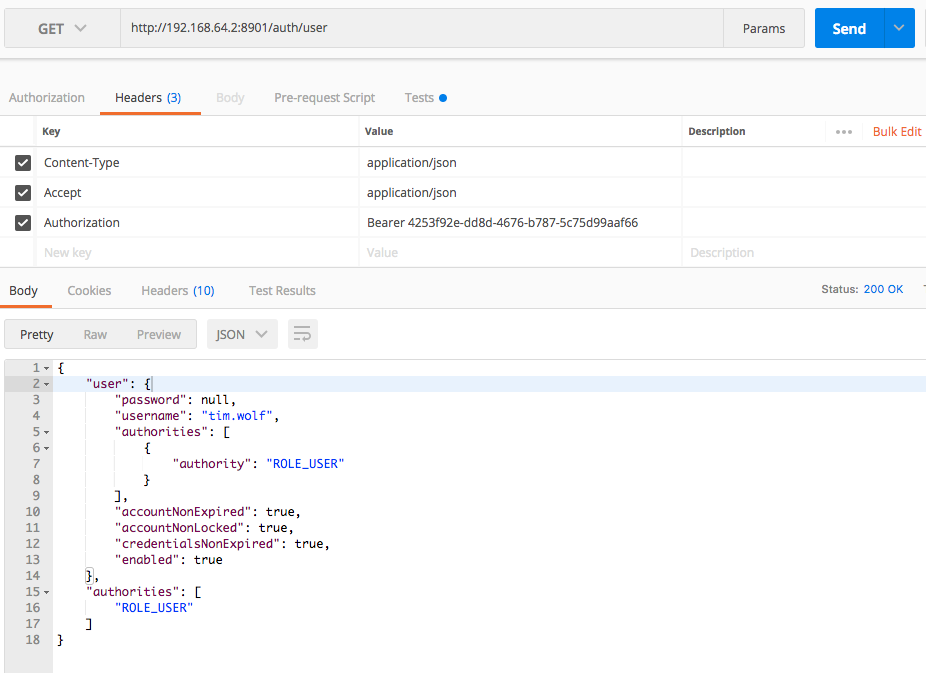
Finally, you perform the POST request and receive the following token:

**"access\_token": "4253f92e-dd8d-4676-b787-5c75d99aaf66"**

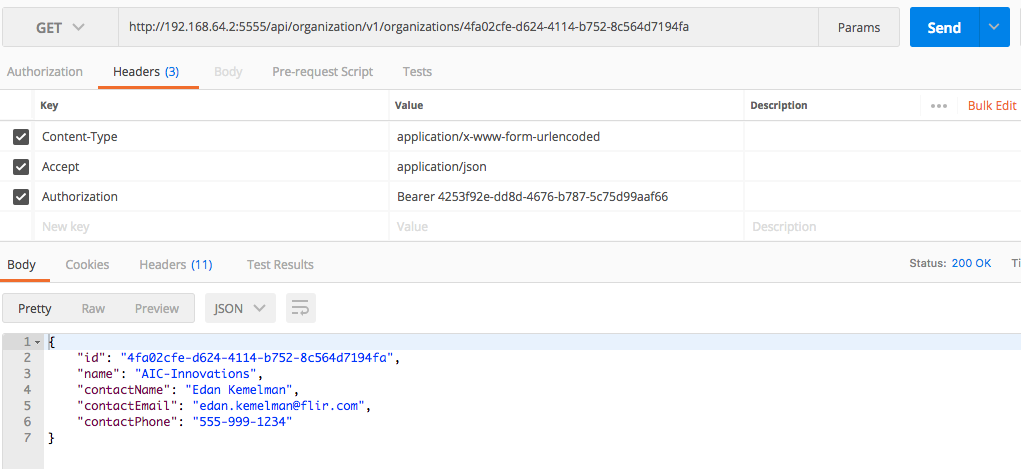


In the next step, you issue an HTTP GET request against the **/auth/user** endpoint , first setting the request Authorization to NoAuth as well as passing this OAuth2 access token in the request Header’s Authorization key , with a value of Bearer **4253f92e-dd8d-4676-b787-5c75d99aaf66** .

Now, upon issuing the GET request we receive the following response :



Finally with the OAuth2 token added to the GET request Header’s Authorization key with value Bearer < OAuth 2 Token> , we issue the same request we initially attempted above (without the token) and confirm that the user **tim.wolf** with ROLE \_USER is able to access the resource:



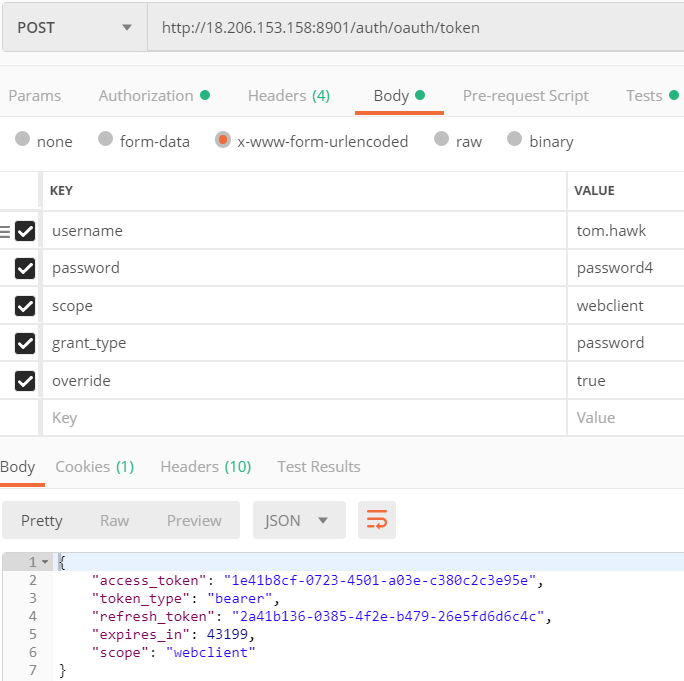
**Use Case 2: Protect a Service via a Specific Role**

In the second use case, you explore how to protect a service via a specific role. In this section we comment the first configure method and uncomment the second implementation of this method in the **organization service’s ResourceServerConfiguration** class.

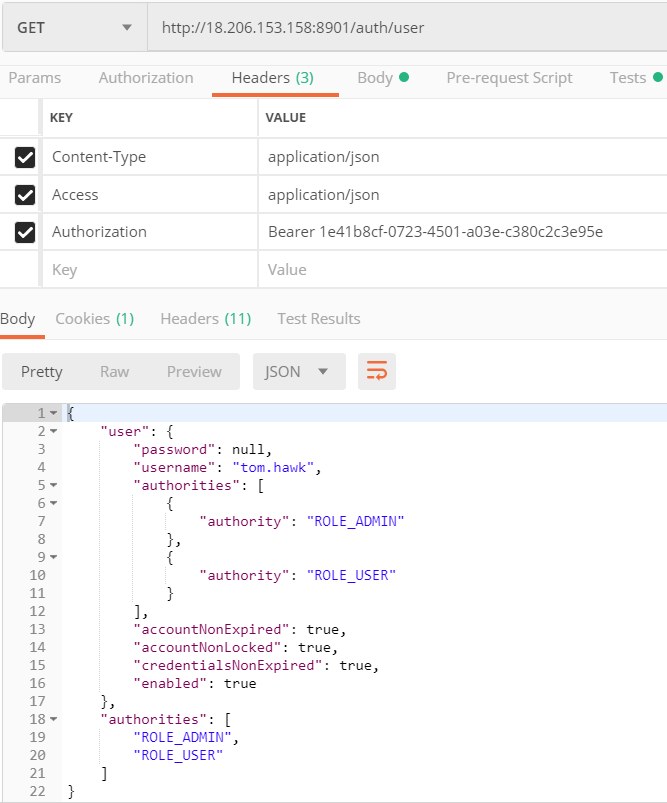
You then rebuild and restart the application. Once up, you create a bearer token for the user **tom.hawk** that we created an account for

with access to EagleEye services with the USER and ADMIN role :

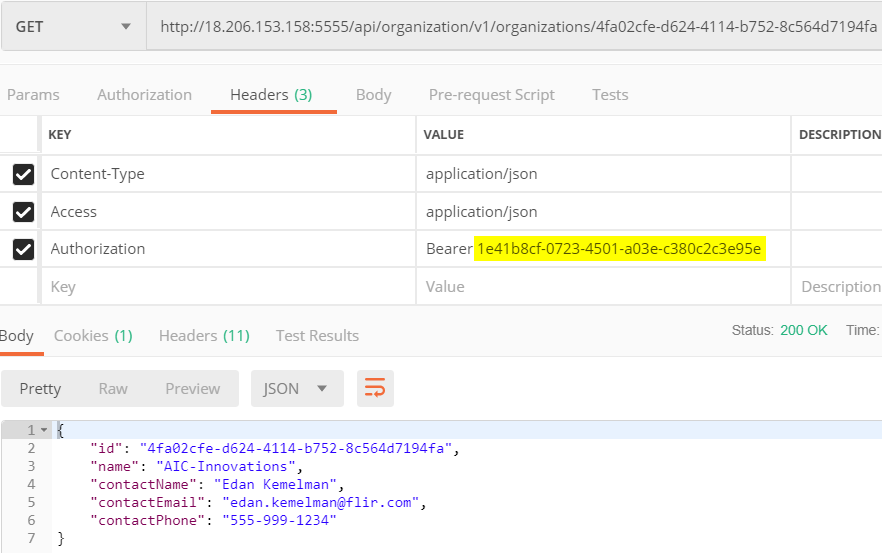
"access\_token": "1e41b8cf - 0723 - 4501 - a03e - c380c2c3e95e"



Upon issuing a GET request against the **/auth/user** endpoint we receive the following response



Now, with the OAuth2 token for tom.hawk added to the GET request Header’s Authorization key with value Bearer <OAuth2Token>, we issue the following request and confirm that the user **tom.hawk** with ROLE\_USER is able to access the resource:



In addition, this user which also has ROLE\_ADMIN can delete the resource

