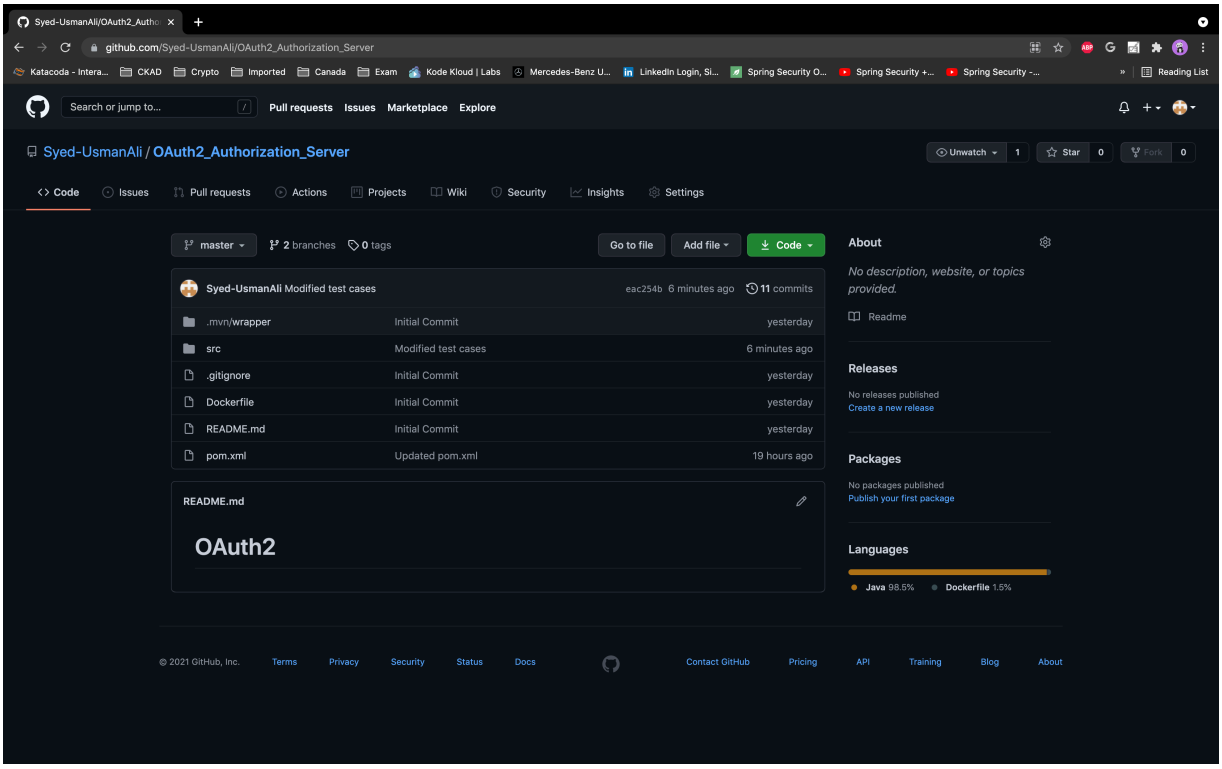
GitHub Repo Details
———————————

https://github.com/Syed-UsmanAli/OAuth2_Authorization_Server

Final code is placed in the default “master” branch and not on “main” branch.

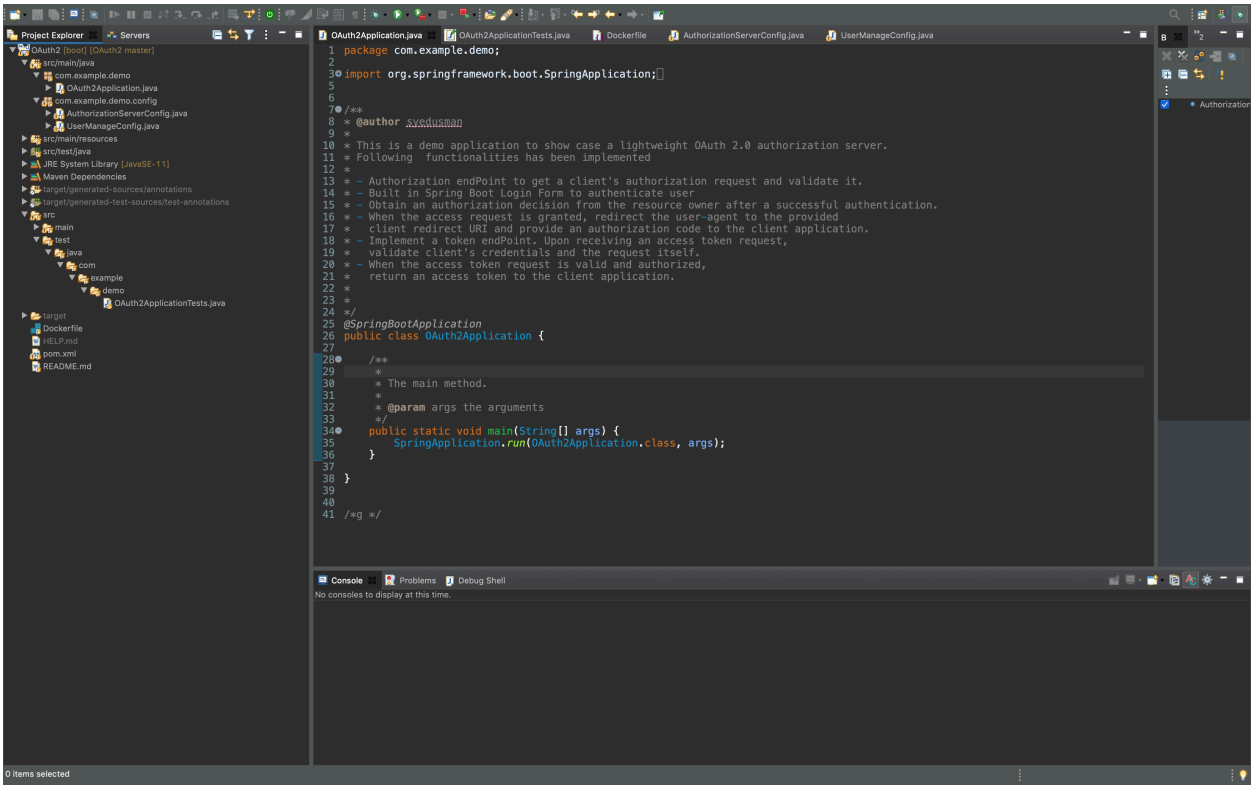


IDE
———

Spring Tool Suite 4 IDE was used and the project has been developed as a Spring Boot Project. Maven is the build tool. The project has also been Dockerized and image has been uploaded on Docker Hub.

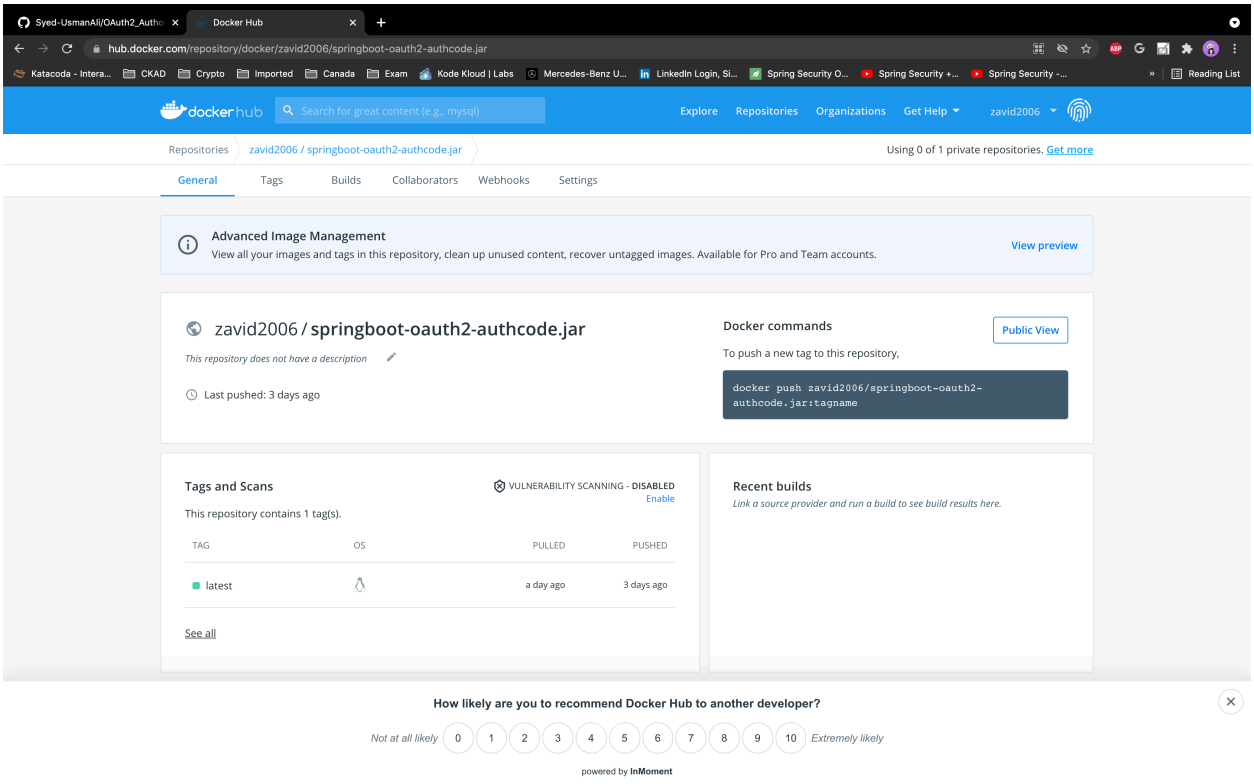
Jenkins was used for CI .Pipeline was configured to trigger maven build on each commit to GIT. It would then build and boot up the Spring Boot Application, followed by executing the Unit test cases placed in the project.

Screenshot of Project Folder Structure inside SpringBoot App

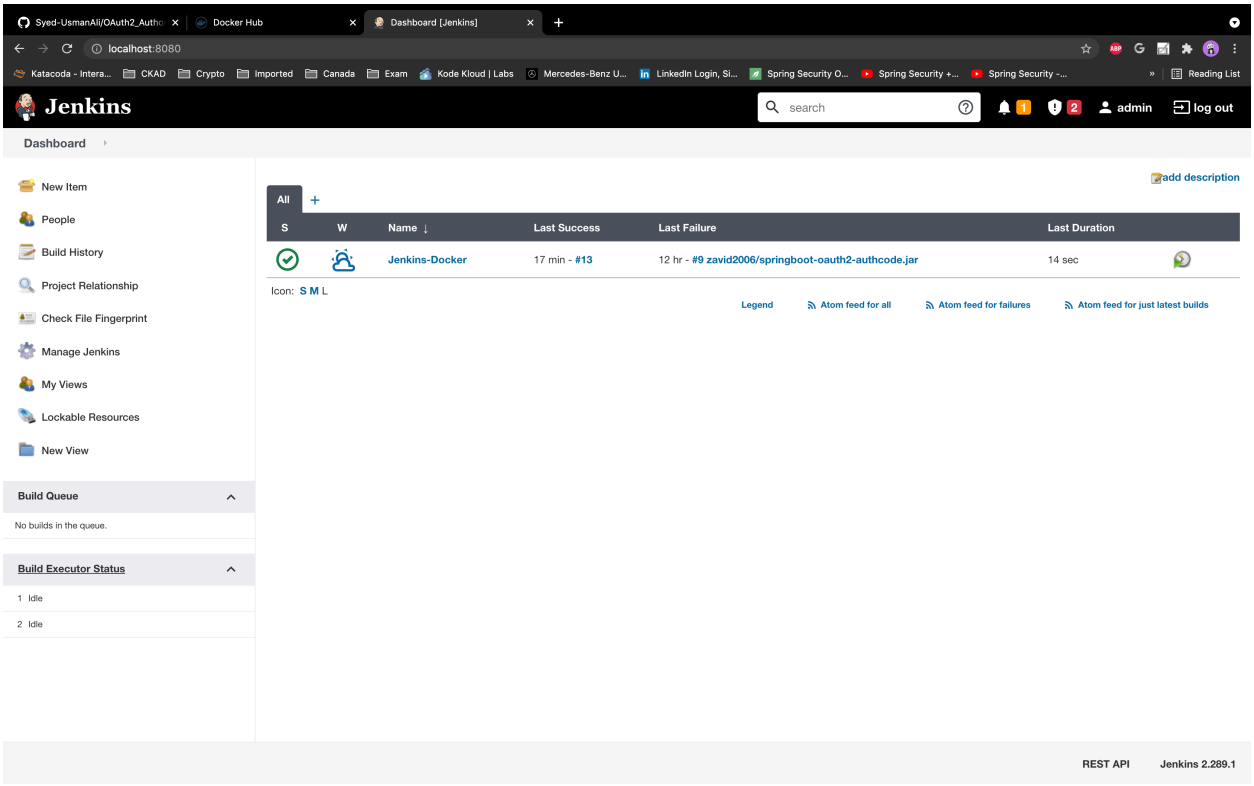


Screenshot of Uploaded Docker Hub Image

https://hub.docker.com/repository/docker/zavid2006/springboot-oauth2-authcode.jar

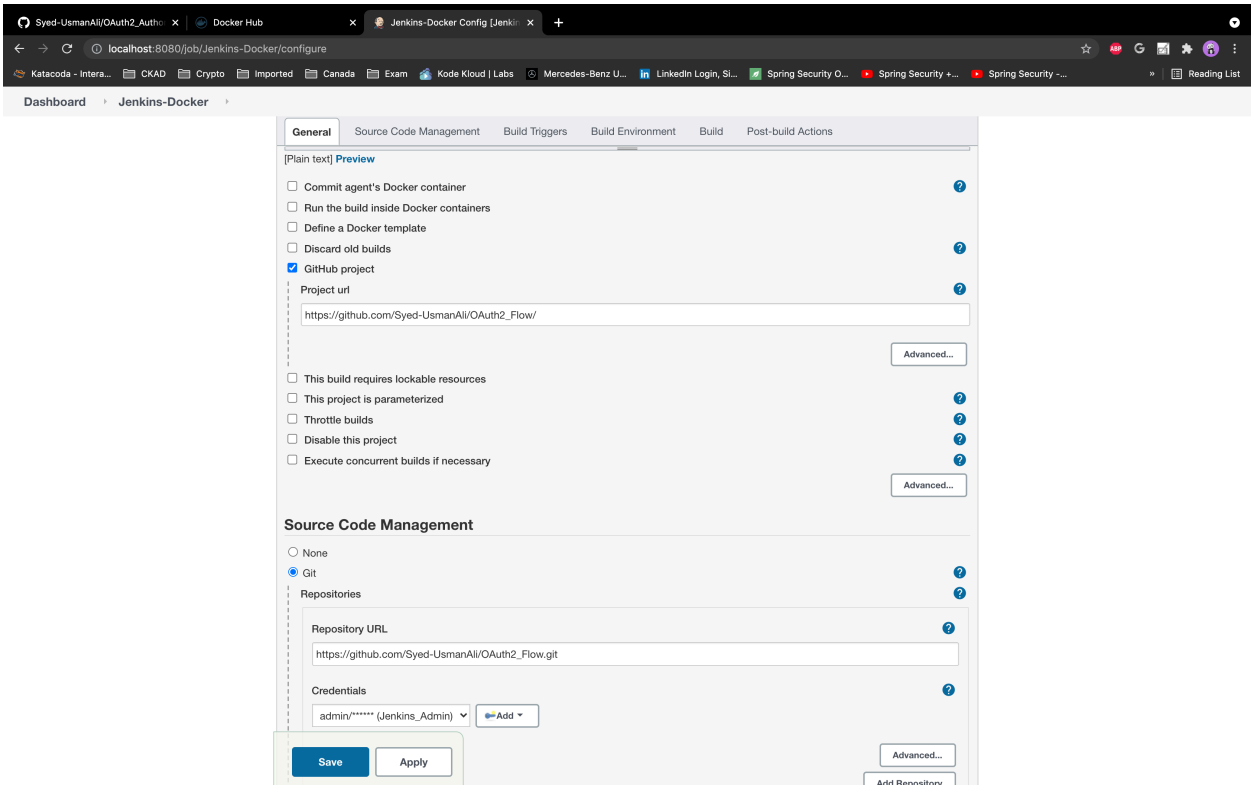


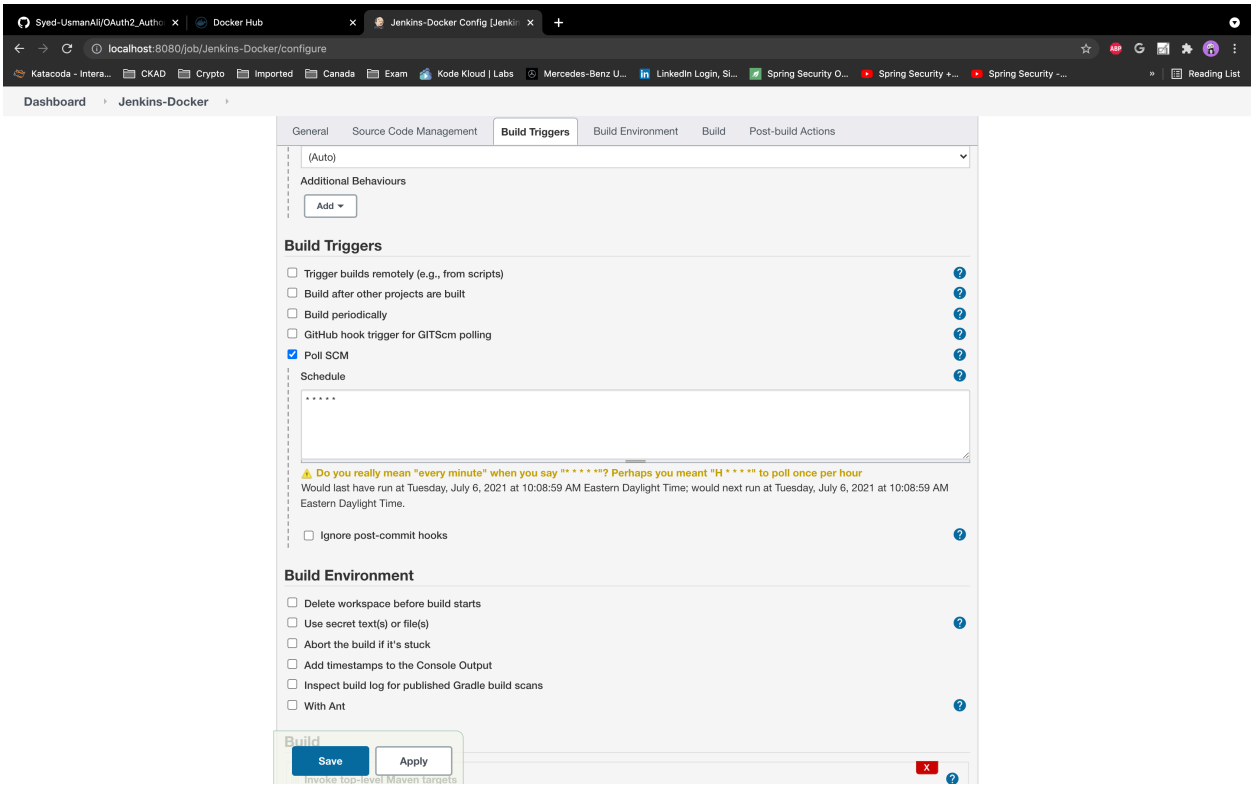
Screenshot of Local Jenkins

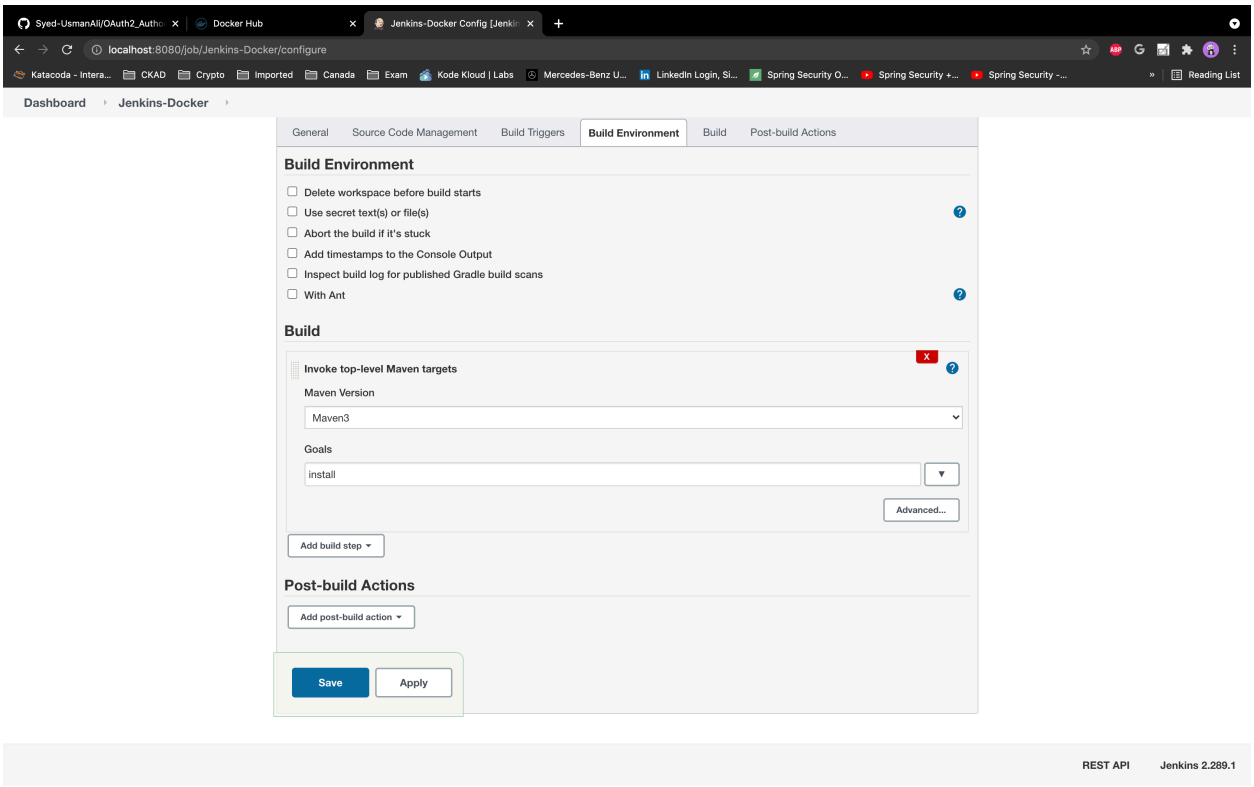


Jenkins Pipeline was configured with

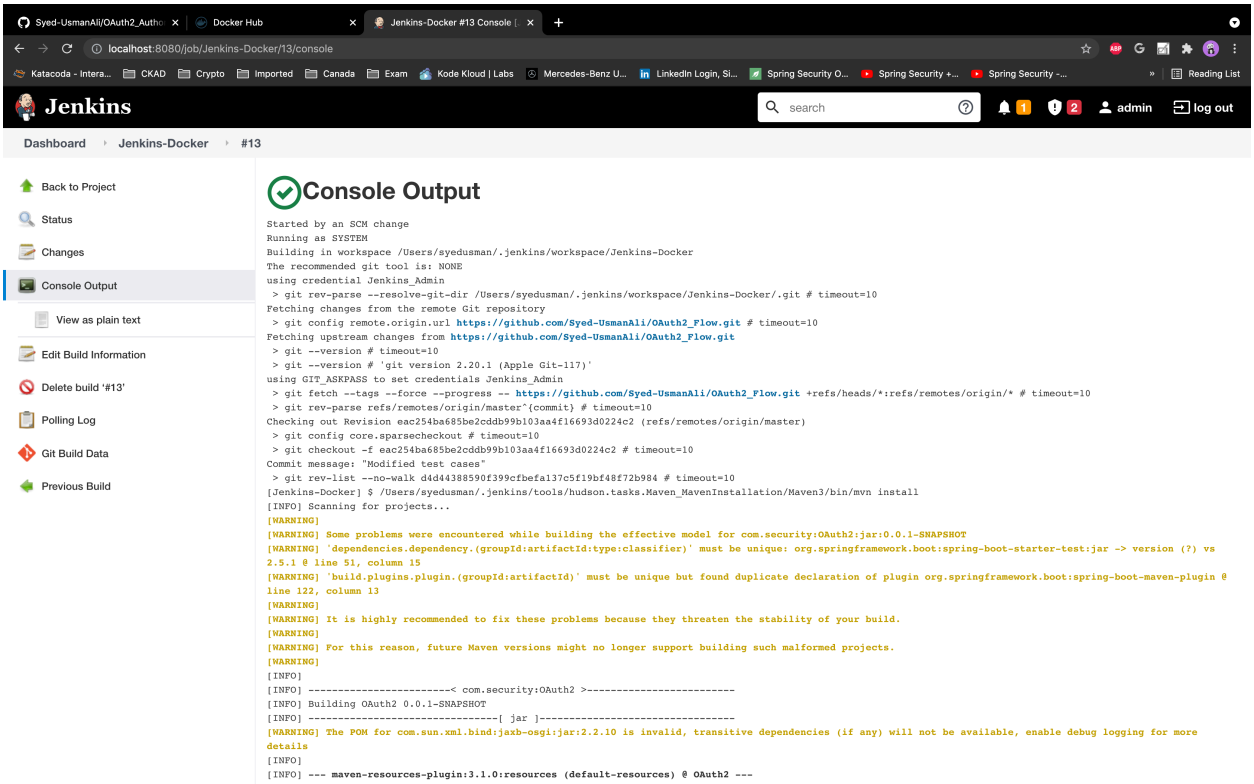
1) GitHub Repo to trigger maven build on target “install”on every commit to “master” branch

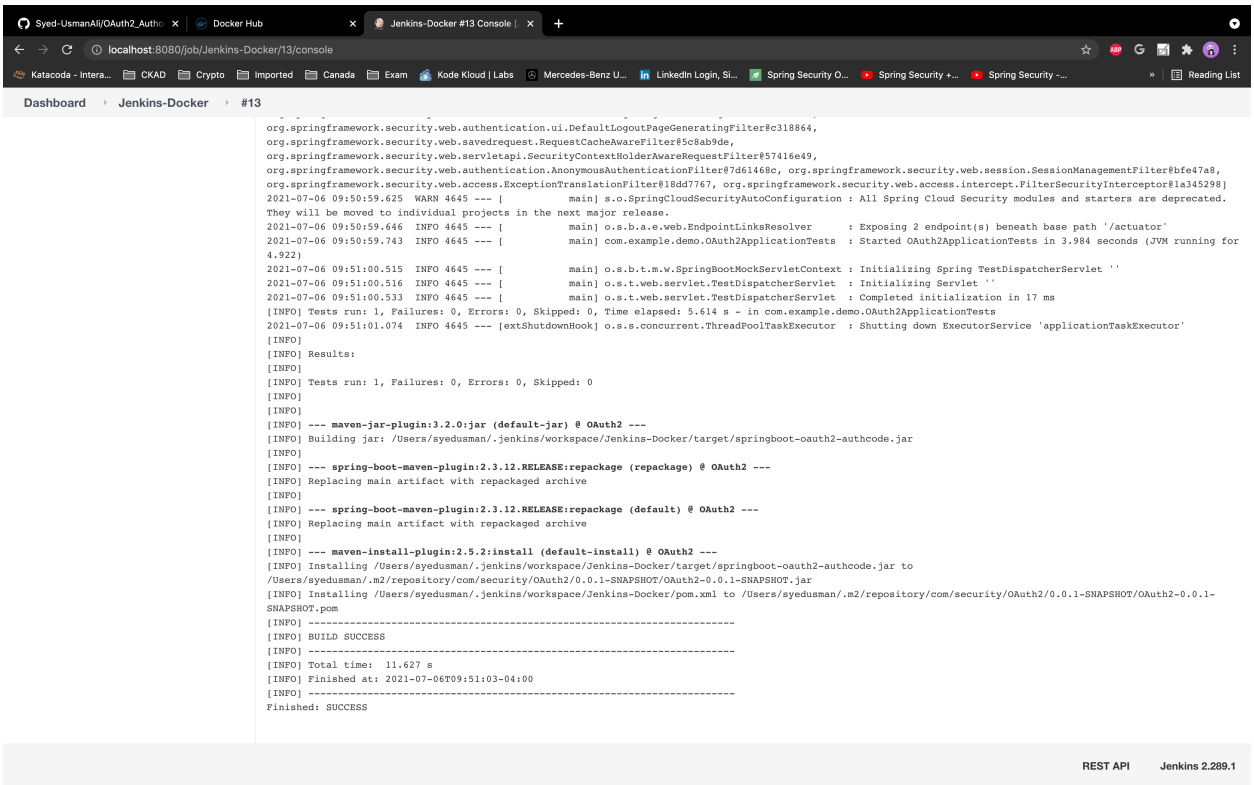



Syed-UsmanAli/OAuth2_Autho
Docker Hub
Jenkins-Docker Config [Jenkin
localhost:8080/job/Jenkins-Docker/configure
Dashboard
Jenkins-Docker
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
(Auto)
Additional Behaviours
Add
Build Triggers
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
GitHub hook trigger for GITScm polling
Poll SCM
Schedule
* * * * *
Do you really mean "every minute" when you say "* * * * *"? Perhaps you meant "H * * * *" to poll once per hour
Would last have run at Tuesday, July 6, 2021 at 10:08:59 AM Eastern Daylight Time; would next run at Tuesday, July 6, 2021 at 10:08:59 AM Eastern Daylight Time.
Ignore post-commit hooks
Build Environment
Delete workspace before build starts
Use secret text(s) or file(s)
Abort the build if it's stuck
Add timestamps to the Console Output
Inspect build log for published Gradle build scans
With Ant
Save
Apply


Syed-UsmanAli/OAuth2_Autho
Docker Hub
Jenkins-Docker Config [Jenkin
localhost:8080/job/Jenkins-Docker/configure
Dashboard
Jenkins-Docker
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Build Environment
Delete workspace before build starts
Use secret text(s) or file(s)
Abort the build if it's stuck
Add timestamps to the Console Output
Inspect build log for published Gradle build scans
With Ant
Build
Invoke top-level Maven targets
Maven Version
Maven3
Goals
install
Advanced...
Add build step
Post-build Actions
Add post-build action
Save
Apply
REST API
Jenkins 2.289.1

One such sample “console output” from Jenkins





Steps to download, setup and run locally to test and validate
————————————————————————————————-

Pre-requisite:

JDK 8 or higher is installed
Docker is installed like Docker for Desktop
Maven is Installed
Any IDE like Spring Tool Suite or IntelliJ or Eclipse is available
Jenkins is Installed
Postman client is installed

There are **2 ways** to start this spring boot project.

A) Download the Code from GIT and import the project in to any IDE , launch as Spring boot App and execute the Junit test case in the project .

Note: Junit Test case has been written only for grant Type=password flow

Or

B) Pull the Docker image from the docker hub and docker run to launch the spring boot app.

Note: Testing through Postman Client is available for both grant Type=password and Grant type= Authorization Code flow.

Pull down the code from the GitHub repo for reference and import it to any IDE as a Spring Boot Project

https://github.com/Syed-UsmanAli/OAuth2_Authorization_Server

Command :
On any directory where the project needs to be setup execute below command

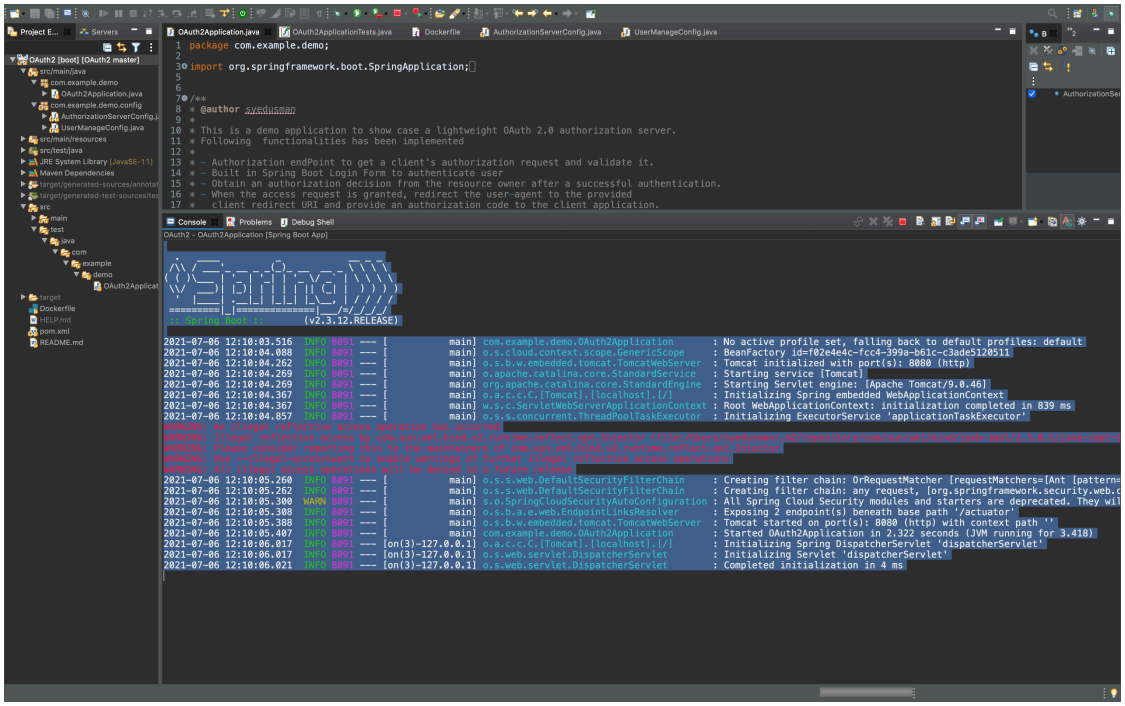
Let's say directory is c:/users/syedusman/OAuth2_Project

In Terminal /Command Prompt execute the below commands , as shown in screenshot.
Verify it is cloned properly with Project source code files, pom.xml and Dockerfile.

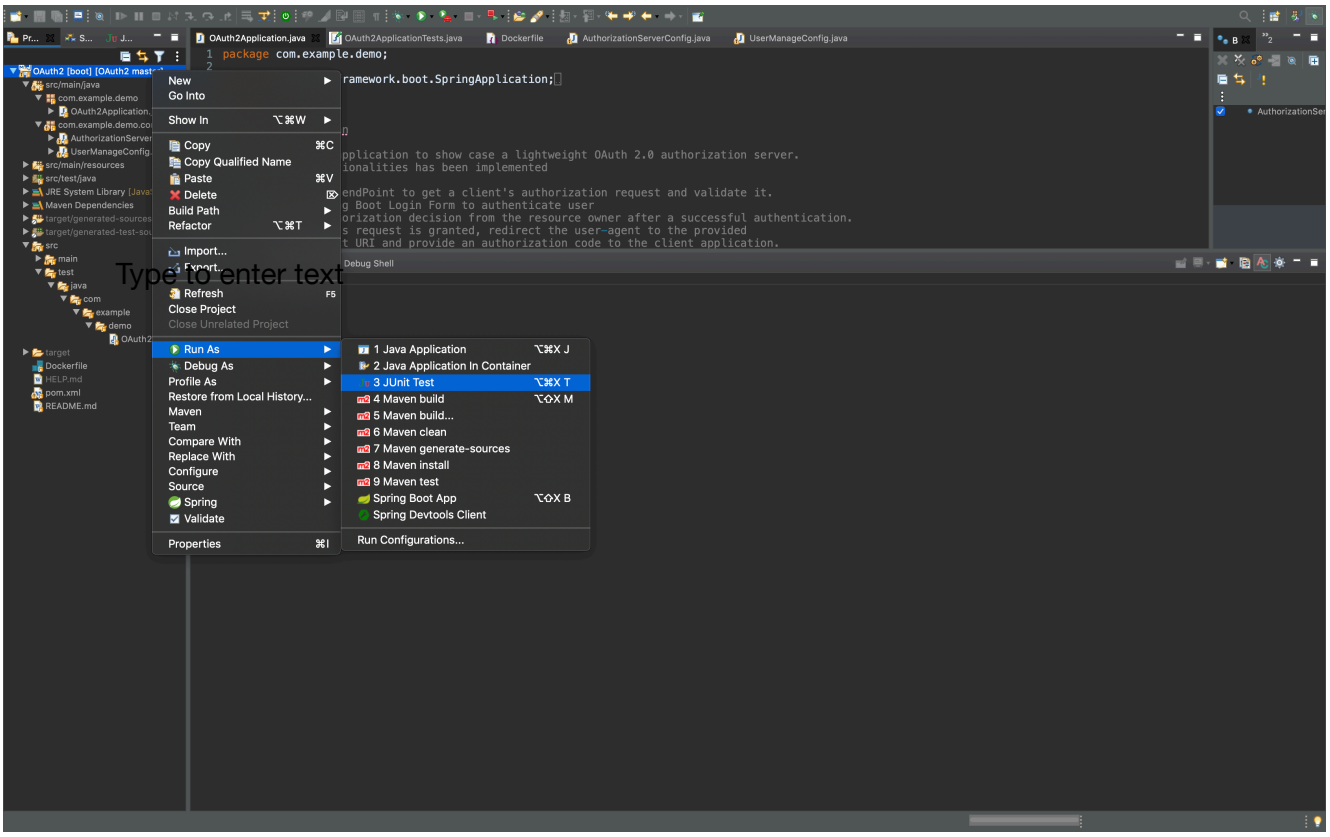
- git init
- git clone https://github.com/Syed-UsmanAli/OAuth2_Authorization_Server.git

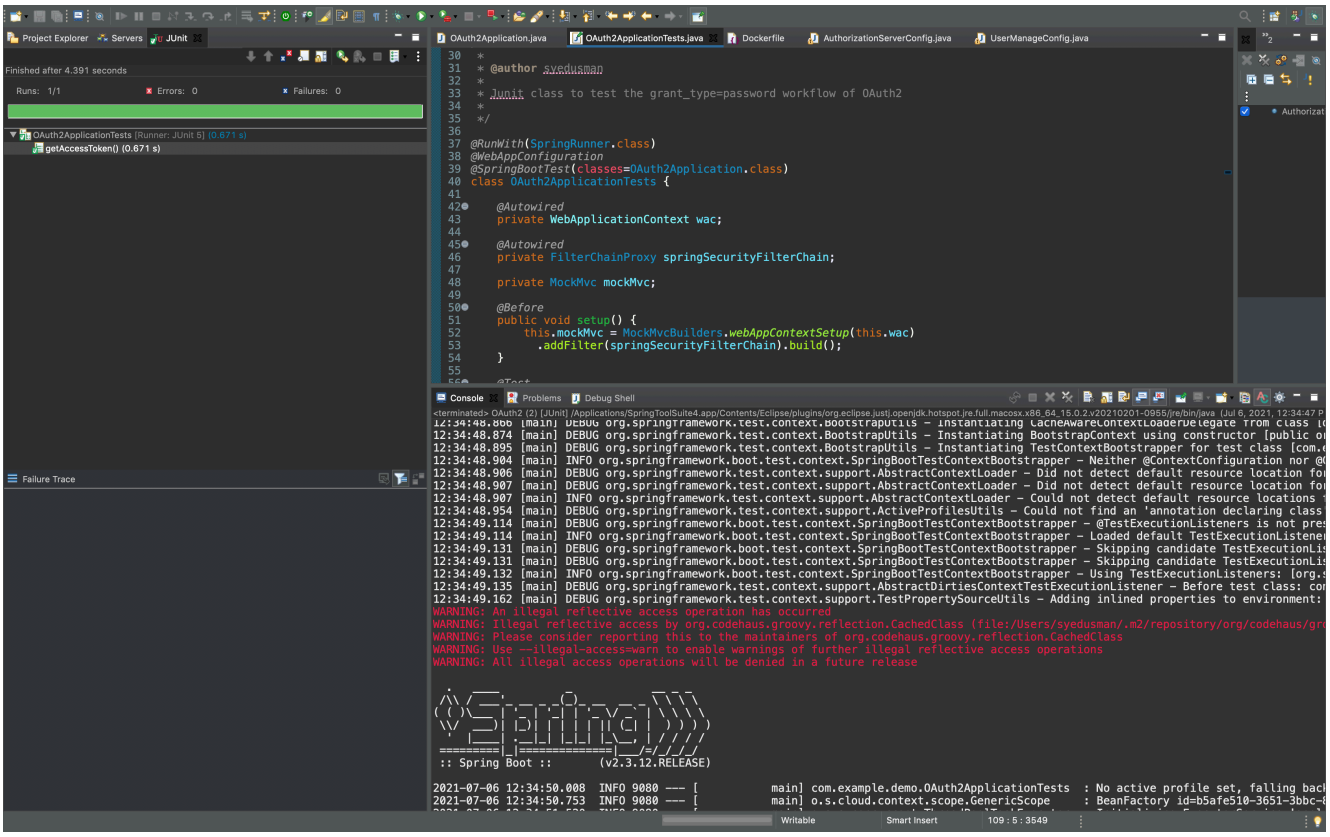
```
Syeds-MacBook-Pro:OAuth2_Project syedusman$ pwd
/users/syedusman/OAuth2_Project
Syeds-MacBook-Pro:OAuth2_Project syedusman$ git init
Initialized empty Git repository in /Users/syedusman/OAuth2_Project/.git/
Syeds-MacBook-Pro:OAuth2_Project syedusman$ git clone https://github.com/Syed-UsmanAli/OAuth2_Authorization_Server.git
Cloning into 'OAuth2_Authorization_Server'...
remote: Enumerating objects: 142, done.
remote: Counting objects: 100% (142/142), done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 142 (delta 32), reused 125 (delta 21), pack-reused 0
Receiving objects: 100% (142/142), 61.00 KiB | 2.54 MiB/s, done.
Resolving deltas: 100% (32/32), done.
Syeds-MacBook-Pro:OAuth2_Project syedusman$ ls
OAuth2_Authorization_Server
Syeds-MacBook-Pro:OAuth2_Project syedusman$ cd OAuth2_Authorization_Server
Syeds-MacBook-Pro:OAuth2_Authorization_Server syedusman$ ls
Dockerfile      README.md       pom.xml         src
Syeds-MacBook-Pro:OAuth2_Authorization_Server syedusman$ ls -lrt
total 32
-rw-r--r--  1 syedusman  staff   161 Jul  6 11:56 Dockerfile
-rw-r--r--  1 syedusman  staff     9 Jul  6 11:56 README.md
-rw-r--r--  1 syedusman  staff  4359 Jul  6 11:56 pom.xml
drwxr-xr-x  4 syedusman  staff   128 Jul  6 11:56 src
Syeds-MacBook-Pro:OAuth2_Authorization_Server syedusman$
```

3) Import the downloaded project in to an IDE, say STS.Right click on Project root directory and run as Spring Boot App



In order to execute Junit test case, stop the Spring boot Server as the Junit will also bring up the spring boot app on the same port 8080 and then run the Test class defined in the project under “src/main/test” package

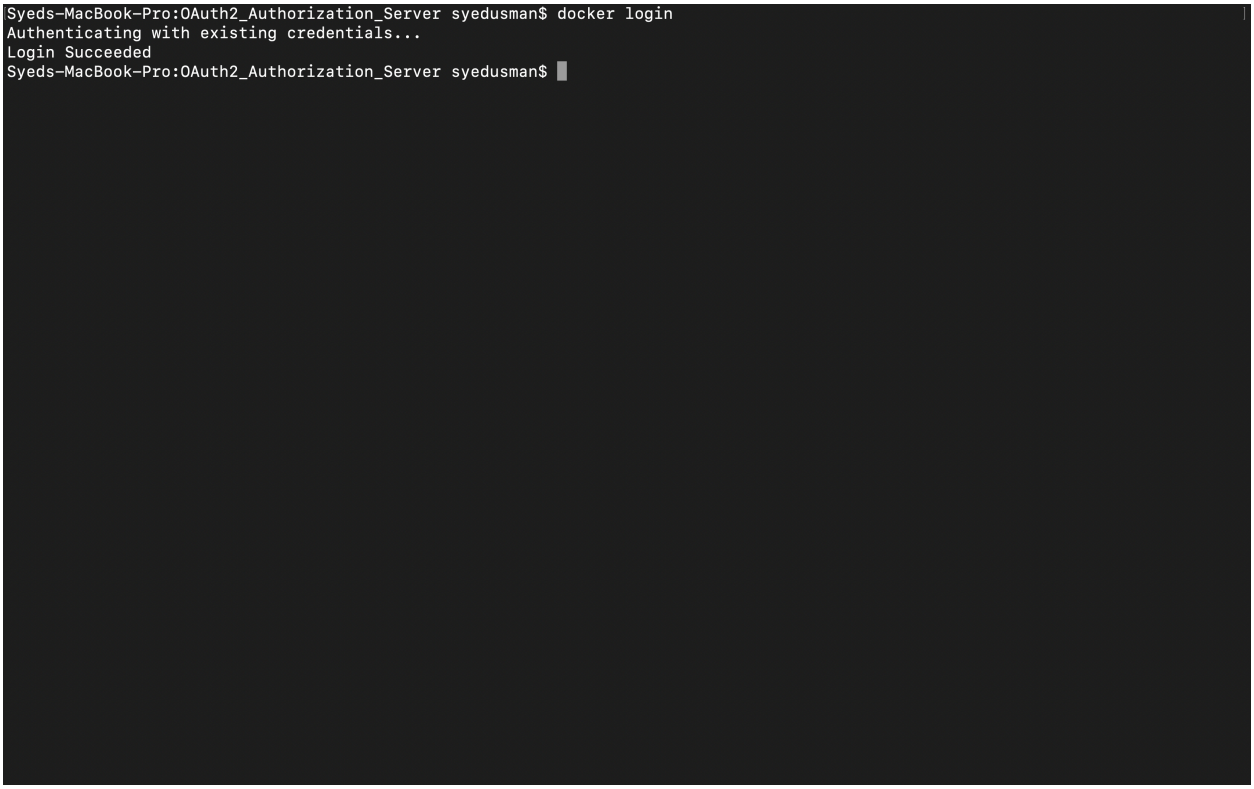

Type to enter text
New
Go Into
Show In
Copy
Copy Qualified Name
Paste
Delete
Build Path
Refactor
Import...
Export...
Refresh
Close Project
Close Unrelated Project
Run As
Debug As
Profile As
Restore from Local History...
Maven
Team
Compare With
Replace With
Configure
Source
Spring
Validate
Properties
1 Java Application
2 Java Application In Container
3 JUnit Test
4 Maven build
5 Maven build...
6 Maven clean
7 Maven generate-sources
8 Maven install
9 Maven test
Spring Boot App
Spring Devtools Client
Run Configurations...

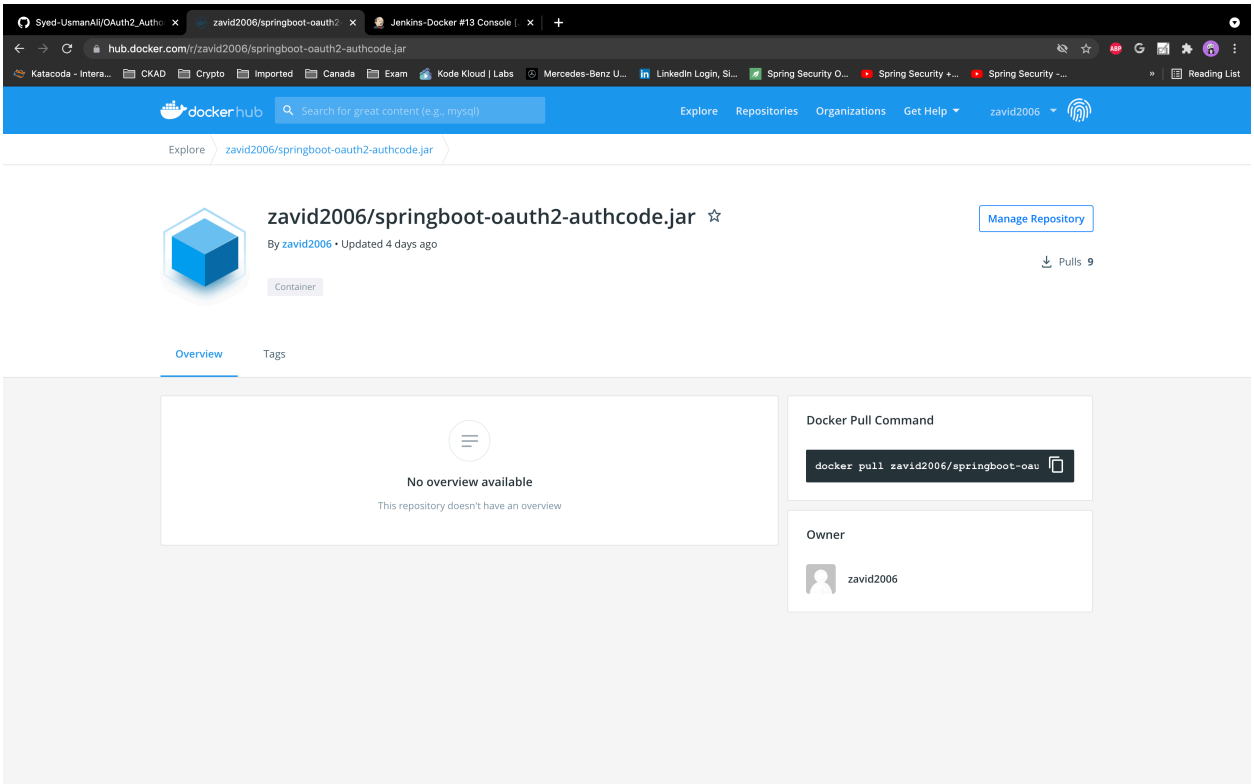

Finished after 4.391 seconds
Runs: 1/1
Errors: 0
Failures: 0
OAuth2ApplicationTests [Runner: JUnit 5] (0.671 s)
getAccessToken() (0.671 s)
Failure Trace
@author syedusman
Junit class to test the grant_type=password workflow of OAuth2
@RunWith(SpringRunner.class)
@WebAppConfiguration
@SpringBootTest(classes=OAuth2Application.class)
class OAuth2ApplicationTests {
@Autowired
private WebApplicationContext wac;
@Autowired
private FilterChainProxy springSecurityFilterChain;
private MockMvc mockMvc;
@Before
public void setup() {
this.mockMvc = MockMvcBuilders.webAppContextSetup(this.wac)
.addFilter(springSecurityFilterChain).build();
}
WARNING: An illegal reflective access operation has occurred
WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.reflection.CachedClass
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
:: Spring Boot :: (v2.3.12.RELEASE)
2021-07-06 12:34:50.008 INFO 9080 --- [main] com.example.demo.OAuth2ApplicationTests : No active profile set, falling back
2021-07-06 12:34:50.753 INFO 9080 --- [main] o.s.cloud.context.scope.GenericScope : BeanFactory id=b5afe510-3651-3bbc-8

Now, we will see the Alternate way of how to pull the docker image from docker hub and run instead of downloading the code from GitHub.

Prerequisite : Docker to be installed and a docker account is available.

1) Execute Docker Login in Terminal / Command Prompt





Pull the Image from docker hub by executing below command in terminal in any directory as shown below

docker pull zavid2006/springboot-oauth2-authcode.jar

```
Syeds-MacBook-Pro:OAuth2Docker syedusman$ pwd
/users/syedusman/OAuth2Docker
Syeds-MacBook-Pro:OAuth2Docker syedusman$ docker login
Authenticating with existing credentials...
Login Succeeded
Syeds-MacBook-Pro:OAuth2Docker syedusman$ docker pull zavid2006/springboot-oauth2-authcode.jar
Using default tag: latest
latest: Pulling from zavid2006/springboot-oauth2-authcode.jar
Digest: sha256:018637de065b7c6abc78e7554d072443772c661f2626f5f2aea7833a906aa6e3
Status: Image is up to date for zavid2006/springboot-oauth2-authcode.jar:latest
docker.io/zavid2006/springboot-oauth2-authcode.jar:latest
Syeds-MacBook-Pro:OAuth2Docker syedusman$
```

Once the docker image is downloaded, execute the below command by running the spring boot application say on the local port 9090

docker run -p 9090:8080 zavid2006/springboot-oauth2-authcode.jar

9090 —> LocalPort
8080 —> Port on which Spring Boot application is up

Once executed, notice below screenshot where the spring boot application launches

```
Syeds-MacBook-Pro:OAuth2Docker syedusman$ pwd
/users/syedusman/OAuth2Docker
Syeds-MacBook-Pro:OAuth2Docker syedusman$ docker login
Authenticating with existing credentials...
Login Succeeded
Syeds-MacBook-Pro:OAuth2Docker syedusman$ docker pull zavid2006/springboot-oauth2-authcode.jar
Using default tag: latest
latest: Pulling from zavid2006/springboot-oauth2-authcode.jar
Digest: sha256:018637de065b7c6abc78e7554d072443772c661f2626f5f2aea7833a906aa6e3
Status: Image is up to date for zavid2006/springboot-oauth2-authcode.jar:latest
docker.io/zavid2006/springboot-oauth2-authcode.jar:latest
Syeds-MacBook-Pro:OAuth2Docker syedusman$ docker run -p 9090:8080 springboot-oauth2-authcode.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::       (v2.3.12.RELEASE)

2021-07-06 16:48:57.986  INFO 1 --- [           main] com.example.demo.OAuth2Application       : No active profile set, falling
back to default profiles: default
2021-07-06 16:49:00.097  INFO 1 --- [           main] o.s.cloud.context.scope.GenericScope     : BeanFactory id=1442c386-8e74-3b
49-8f24-9b1d07547fb8
2021-07-06 16:49:00.732  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s)
: 8080 (http)
2021-07-06 16:49:00.755  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-07-06 16:49:00.756  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apach
e Tomcat/9.0.46]
2021-07-06 16:49:00.905  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded We
bApplicationContext
2021-07-06 16:49:00.906  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: ini
tialization completed in 2862 ms
```

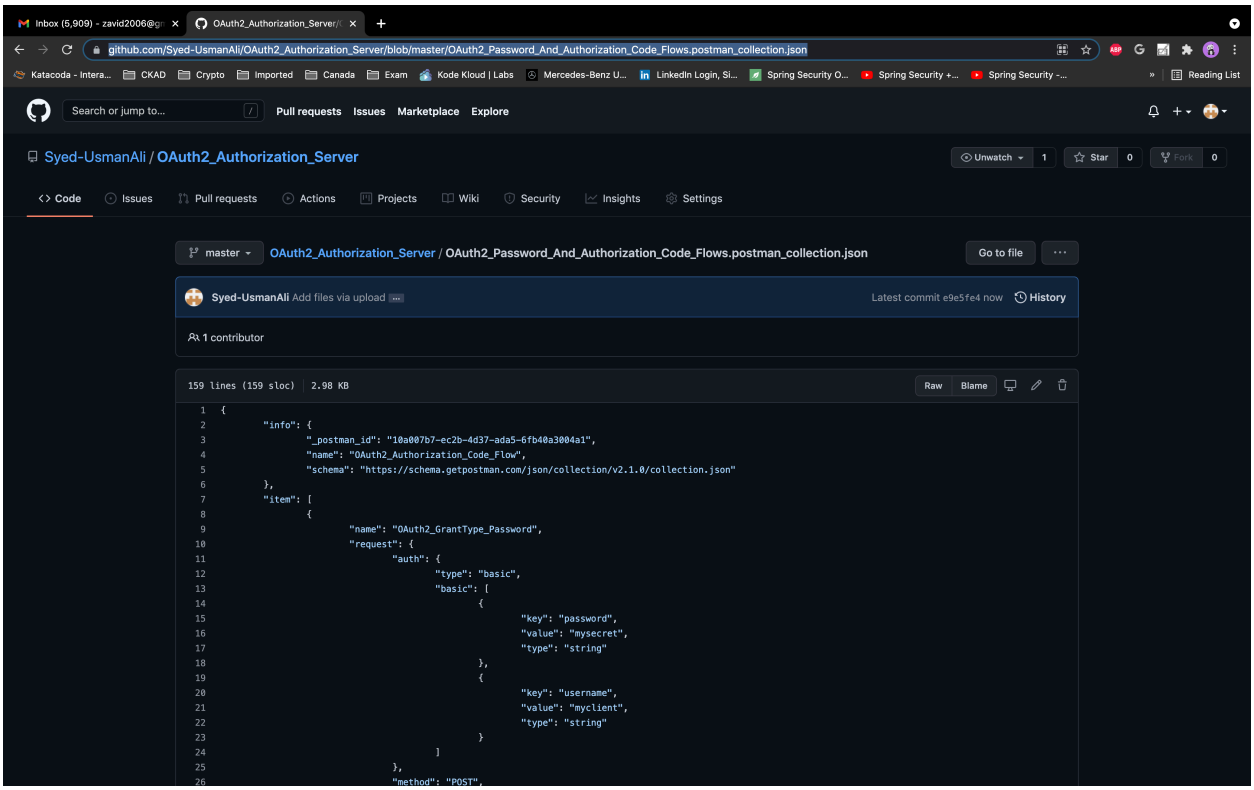
Application is now ready to accept request as shown below.

```
2021-07-06 16:49:00.906  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: ini
tialization completed in 2862 ms
2021-07-06 16:49:02.159  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'a
pplicationTaskExecutor'
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.sun.xml.bind.v2.runtime.reflect.opt.Injector (jar:file:/springboot-oauth2-authcode.jar
!/BOOT-INF/lib/jaxb-impl-2.3.0.1.jar!/) to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int)
WARNING: Please consider reporting this to the maintainers of com.sun.xml.bind.v2.runtime.reflect.opt.Injector
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2021-07-06 16:49:03.385  INFO 1 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: OrReques
tMatcher [requestMatchers=[Ant [pattern='/oauth/token'], Ant [pattern='/oauth/token_key'], Ant [pattern='/oauth/check_token']]],
 [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@78d50a3c, org.springframework.security
.web.context.SecurityContextPersistenceFilter@88a8218, org.springframework.security.web.header.HeaderWriterFilter@44d70181, org.
springframework.security.web.authentication.logout.LogoutFilter@26f1249d, org.springframework.security.web.authentication.www.Ba
sicAuthenticationFilter@4943defe, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@4163f1cd, org.springfram
ework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@23aae55, org.springframework.security.web.authentication.A
nonymousAuthenticationFilter@2fb69ff6, org.springframework.security.web.session.SessionManagementFilter@23c650a3, org.springfram
ework.security.web.access.ExceptionTranslationFilter@18ca3c62, org.springframework.security.web.access.intercept.FilterSecurityI
nterceptor@6df20ade]
2021-07-06 16:49:03.394  INFO 1 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: any requ
est, [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@5ba745bc, org.springframework.secu
rity.web.context.SecurityContextPersistenceFilter@5d10455d, org.springframework.security.web.header.HeaderWriterFilter@2617f816,
 org.springframework.security.web.csrf.CsrfFilter@27f3b6d6, org.springframework.security.web.authentication.logout.LogoutFilter@
5e63cad, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@6759f091, org.springframework.secu
rity.web.authentication.ui.DefaultLoginPageGeneratingFilter@55b62629, org.springframework.security.web.authentication.ui.Default
LogoutPageGeneratingFilter@654b72c0, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@535b8c24, org.springf
ramework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@4a951911, org.springframework.security.web.authenticati
on.AnonymousAuthenticationFilter@55b5e331, org.springframework.security.web.session.SessionManagementFilter@676f0a60, org.spring
framework.security.web.access.ExceptionTranslationFilter@757f675c, org.springframework.security.web.access.intercept.FilterSecur
ityInterceptor@867ba60]
2021-07-06 16:49:03.518  WARN 1 --- [           main] s.o.SpringCloudSecurityAutoConfiguration : All Spring Cloud Security modul
es and starters are deprecated. They will be moved to individual projects in the next major release.
2021-07-06 16:49:03.542  INFO 1 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 2 endpoint(s) beneath
base path '/actuator'
2021-07-06 16:49:03.675  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080
 (http) with context path ''
2021-07-06 16:49:03.705  INFO 1 --- [           main] com.example.demo.OAuth2Application       : Started OAuth2Application in 7.
532 seconds (JVM running for 8.769)
```

TESTING through Postman Client

Note : Postman collection is uploaded to GitHub Repo.Please download and import those requests in to your postman client for local testing

https://github.com/Syed-UsmanAli/OAuth2_Authorization_Server/blob/master/OAuth2_Password_And_Authorization_Code_Flows.postman_collection.json



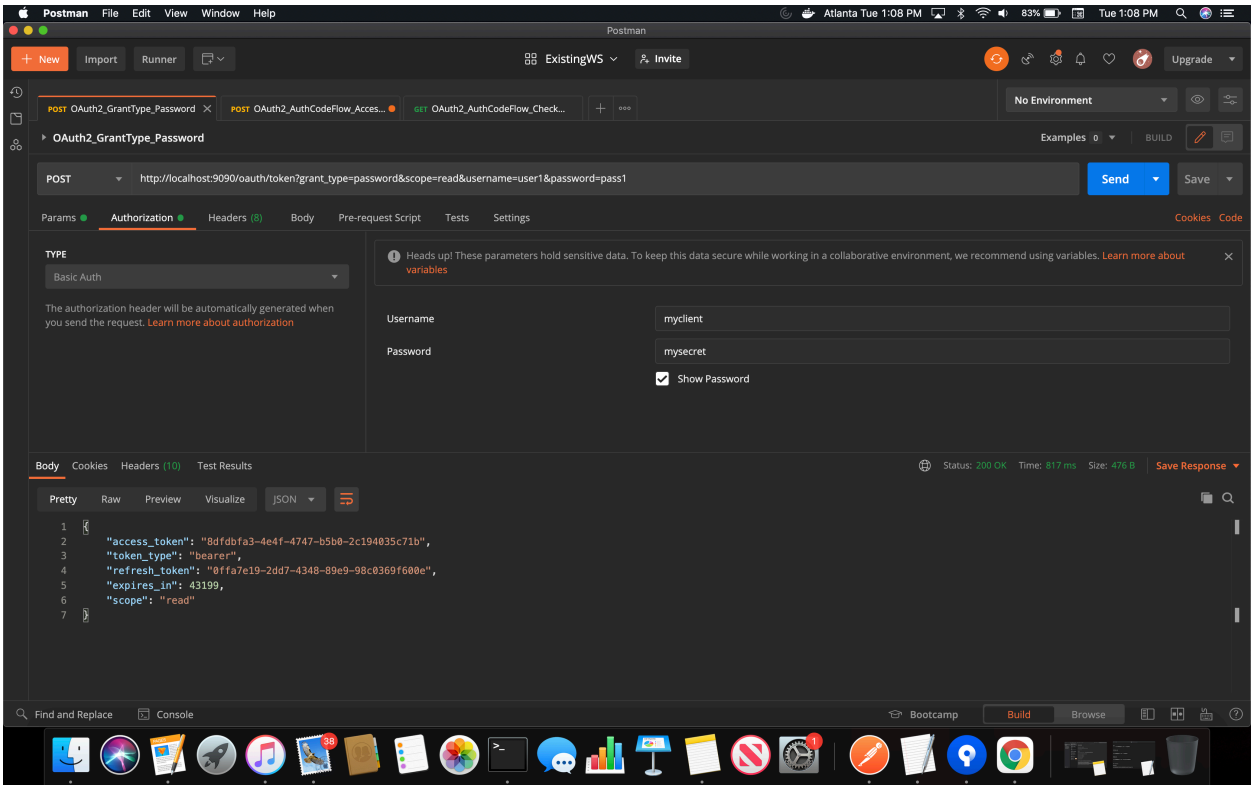
TEST 1 - GrantType = Password Flow

Authorization End Point - /oauth/token

Query Parameters - grant_type=password&scope=read&username=user1&password=pass1

Authorization Header - Basic Auth , username is Client ID and password is ClientSecret which in this case are ‘myclient’ & ‘mysecret'

On Submitting the Post request, In the response, as shown in above screenshot



"**Access Token**" and "**Refresh Token"** are generated with an expiry time.

This "Access Token" can then be used by the Client Application to access resource from the "Resource Server".

TEST 2 - GrantType = Authorization Code

In this flow, first "**Authorization Code**" needs to be obtained by the client before, sending it to get the "**Access Token**" in order to access the Resource from Resource Server.

Test Steps:

A) Open any web browser, and type in the below

http://localhost:9090/oauth/authorize?response_type=code&client_id=myappclient&scope=read

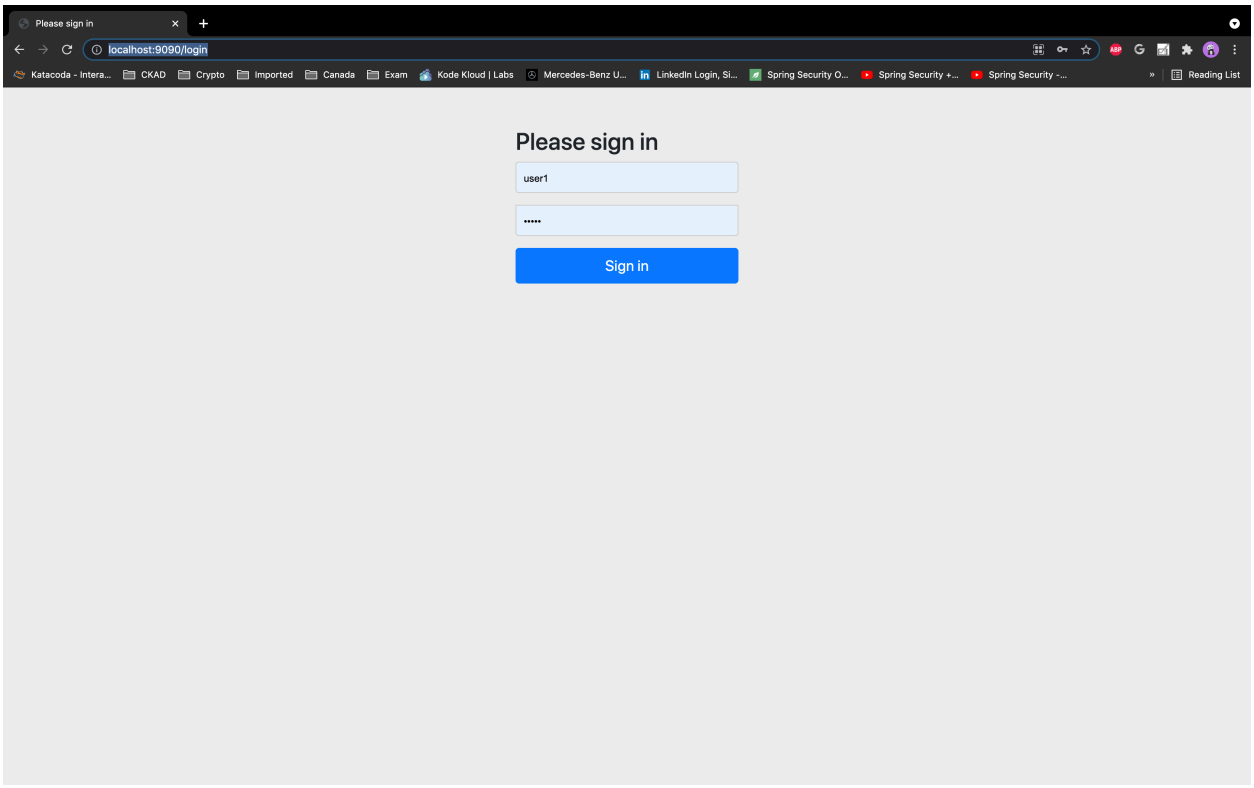
Note: 9090 - Port on which we exposed the spring boot application to run locally

Response type = Code (Represents the Authorization Code Flow)

Client ID - ID of the client application
Scope - What scopes are made available on the resources

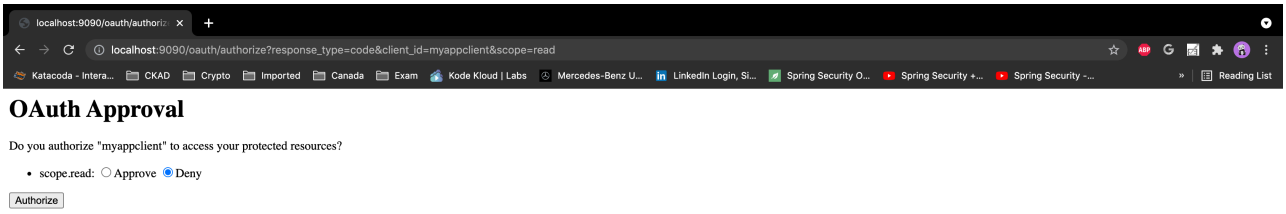
On loading the url, it launches the default Spring Boot Login Form , in order for the user to enter his login credentials as shown below.

http://localhost:9090/login



On Click of Sign In, it navigates to the **/oauth/authorize** **OAuth Approval Page** where it requests for **user consent** to authorize whether the **client** "**myappclient**" is allowed to access the **protected resource** on the **scopes** defined

Once the user selects "**Approve**" & clicks on "**Authorize**" button , the auth code is sent back to the "**Redirect URI**" defined. Since this



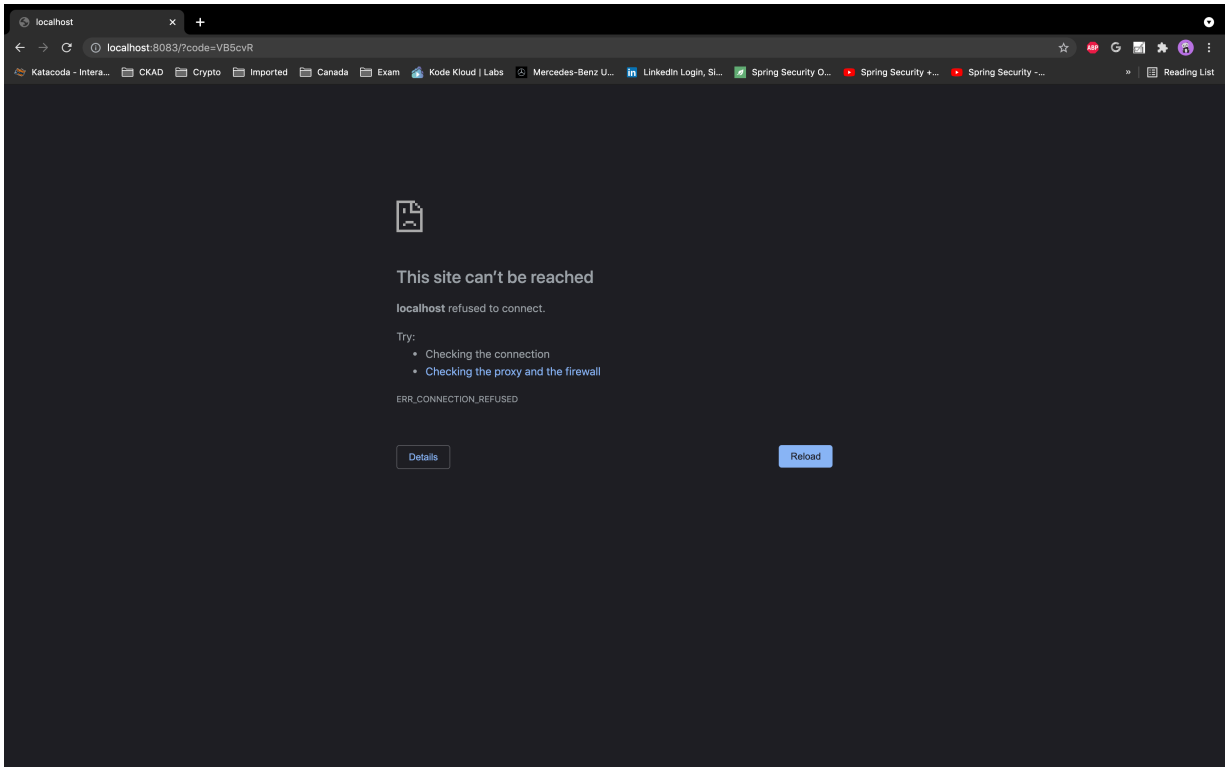
is for demo purpose only, the redirect uri is defined as **localhost** on a different port in order to receive the **auth code** as shown below.

Notice the **auth code generated on the browser url** , example in this case, codes generated as **code=VB5cvR** and ignore the site not reached. In real world applications, it will be redirected to the client applications

Now pass this Auth Code to the **/oauth/token** to get the “Access Token” through Postman Client as shown below

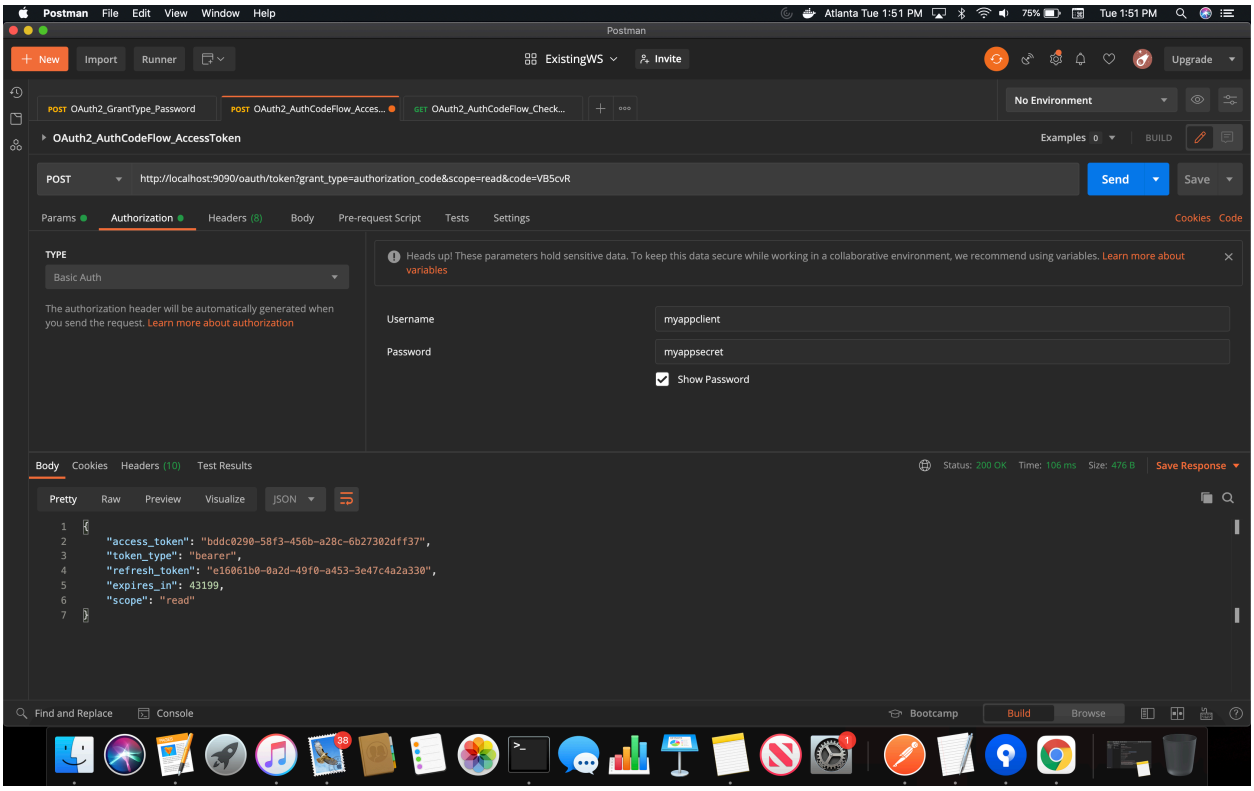
http://localhost:9090/oauth/token?grant_type=authorization_code&scope=read&code=VB5cvR

Note :
Grant Type is “Authorization Code”



Code is the generated code value from the previous /oauth/authorize flow.

Note : In this example, since the port is exposed on 9090, in the **postman** request, it is mentioned as localhost:**9090,** please change it accordingly as per the port that you would expose on the docker run command that was executed earlier.



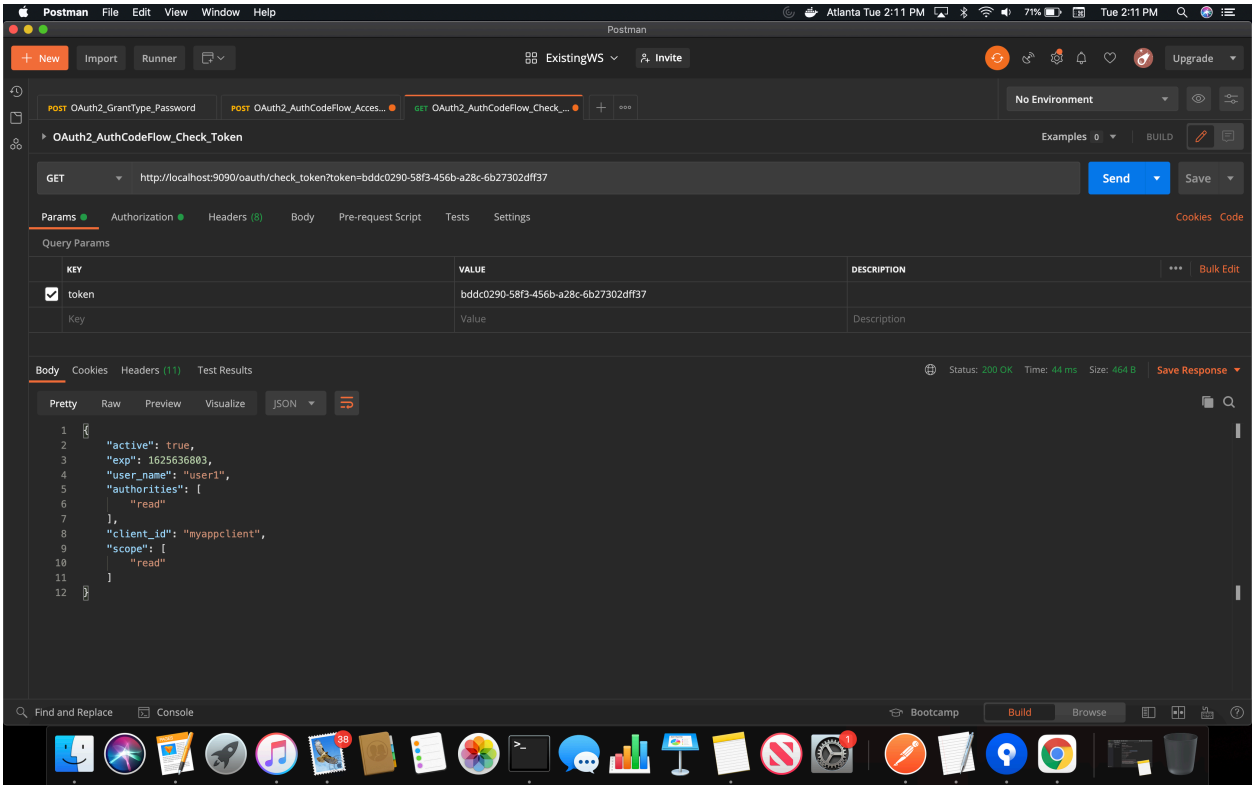
“Access Token” is successfully generated!!! for the **“OAuth2 - Authorization code”** flow

Now, using this “Access Token”, the client application can access the needed resource from the “Resource Server”.

TEST 3 - Check Access Token

In this Test case, validity of the “Access Token” that was generated in the previous step is checked by accessing the url /oauth/check_token

As shown in the below screenshot, since the “Access Token” is still valid, it would respond the “user” info along with the “scope” defined.



In Summary, we have implemented the “**Authorization Serv**er”, for both types of **Grant Type Flow - “password” and “authorization code**” to generate a valid “**Access Token**”. Also “**Refresh Toke**n” is generated and the validity of the Access token is checked through check token.Using the generated “Access Token”, the needed resource can then be accessed from the “Resource Server”.