Stage 4:

Topics learnt in stage 4

* Spring REST
* Introduction to AWS – EC2, RDS, Dynommo DB
* Spring Microservices
* AWS – CI/CD, Apache Kafka and other services

Spring ReSTful Webservices

ReST stands for Representational State Transfer

What is a webservice

Webservices make 2 applications written in different languages to communicate with each other

Usually webservices shares the data in one common format like JSON/XML that end applications will convert to respective language it is written in.

Types of Webservices

There are two types of webservices

1. ReST: It is a new approach uses XML/JSON/CSV/Text and so on for structuring the data, it can intercommunicate with mobile devices, enterprise applications, web applications, desktop applications and so on
2. SOAP: It is an old approach uses only XML, it can intercommunicate with only enterprise applications.

How to create ReSTful webservices

We need to use Spring Framework to create ReSTful webservices, we must have knowledge on

* Spring Data JPA or DAO layer
* Spring Boot
* MVC pattern
* Spring annotations & Dependency Injection
* Eclipse IDE

Topics Remaining:

Exception Handling

Spring Security

Revision on Spring Boot Concepts

Spring Boot & Spring Data JPA with H2 database

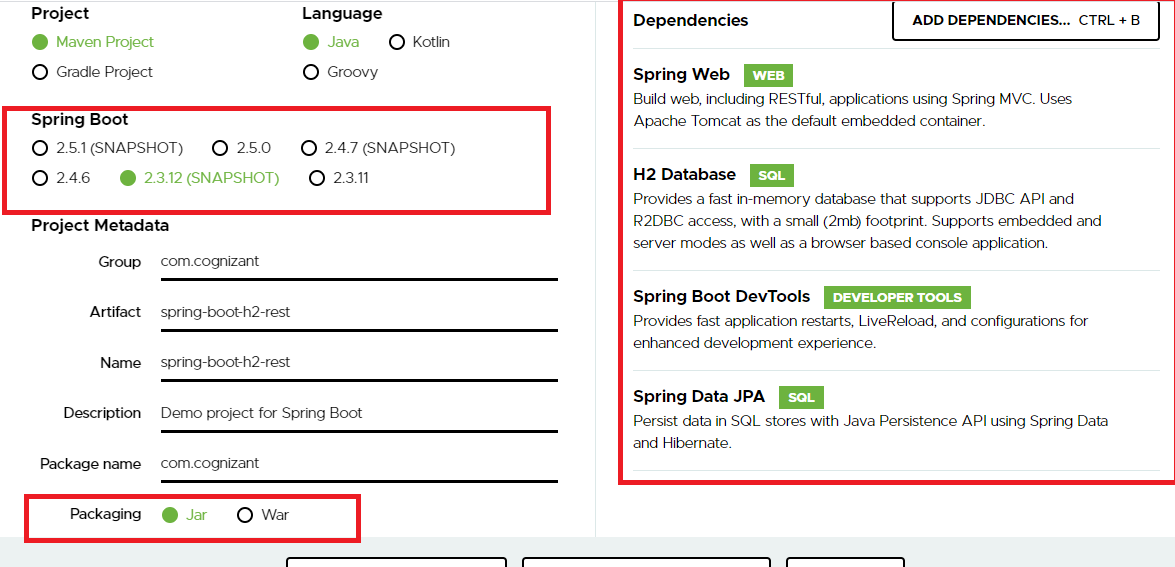
H2 database is an in-memory database which is removed once the application restarts, it is mainly used at the time of development & testing.

Since Spring Boot provides you the feature which separates application code & configurations at the time of development, testing & deployment the changes made in the application.properties would not affect the application logic.

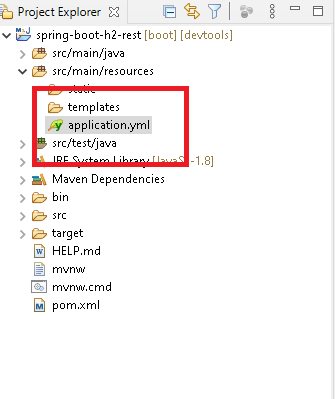
Dependencies required for spring boot & spring data jpa with RESTful webservices

1. Web
2. Devtools
3. H2 database
4. JPA

Spring Initialzr



Change the .properties to .yml so that you can avoid writing repeated properties



Since we are connecting to the database we need to have entities, service, service interface & repositories and also we need to configure the application.yml

application.yml

server:

port: 8081

servlet:

context-path: /api

spring:

datasource:

driver-class-name: org.h2.Driver

url: jdbc:h2:mem:mydb

username: admin

password: admin

Note: Since we are using in-memory database username & password can be anything including database name also

Employee.java

**package** com.cognizant;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

@Entity

**public** **class** Employee {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** employeeId;

**private** String name;

**private** **double** salary;

**public** **int** getEmployeeId() {

**return** employeeId;

}

**public** **void** setEmployeeId(**int** employeeId) {

**this**.employeeId = employeeId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

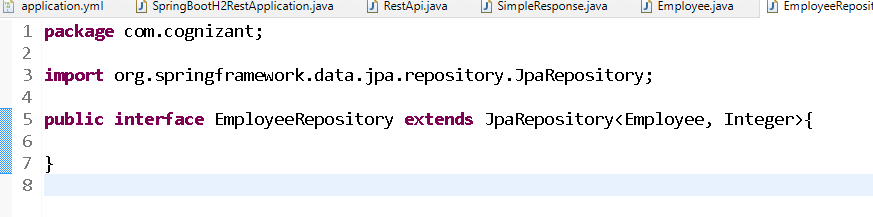
**this**.salary = salary;

}

}

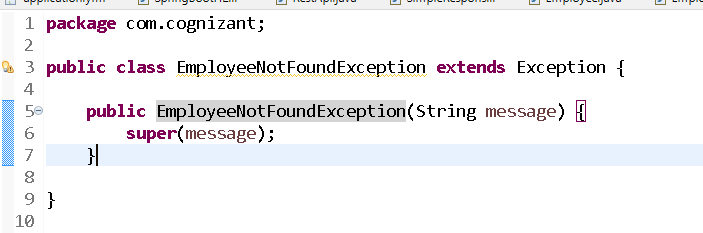
Creating the repository to perform the CURD operations

EmployeeRepository.java



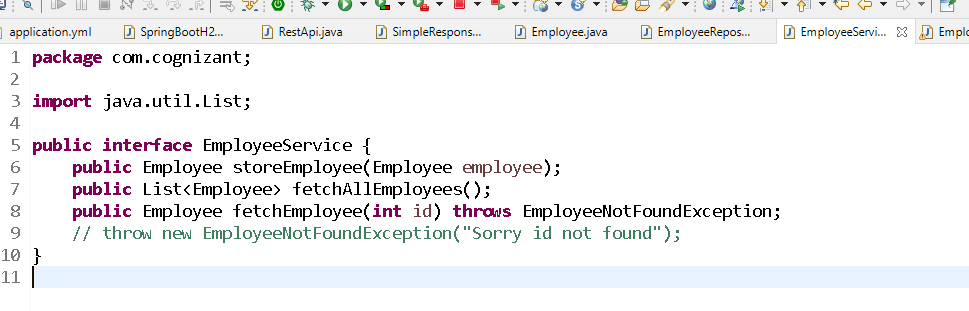
Create one customized exception so that you can throw the exception when necessary

EmployeeNotFoundException.java



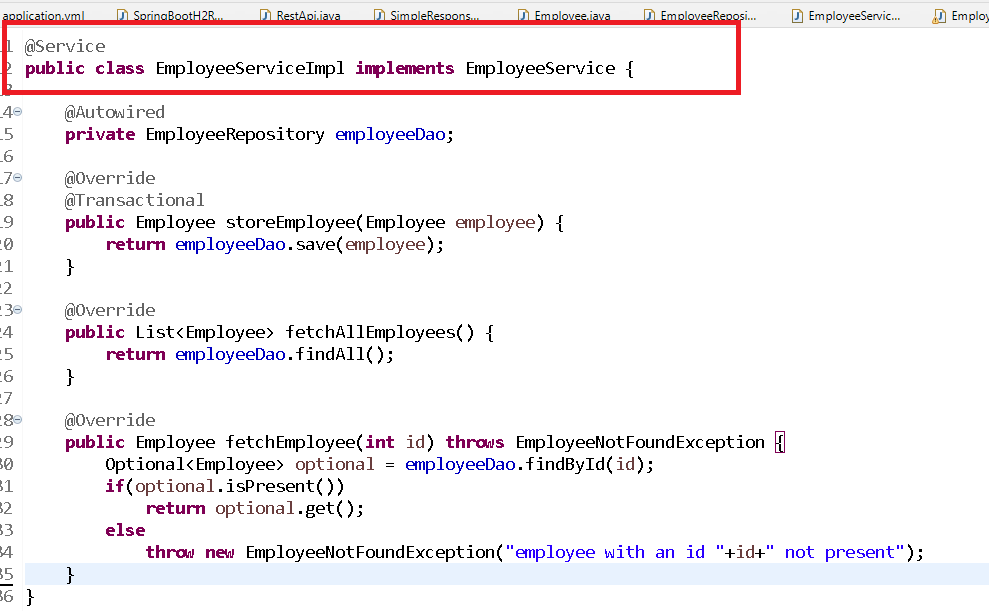
Implementing the service with the help of interface

EmployeeService.java



Implementing the interface

EmployeeServiceImpl.java



Now the @RestController classes call the service methods

**package** com.cognizant;

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

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.MediaType;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("employees")

**public** **class** RestApi {

@Autowired

**private** EmployeeService service;

@PostMapping(consumes = MediaType.***APPLICATION\_JSON\_VALUE***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<Object> storeApi(@RequestBody Employee emp) {

**return** ResponseEntity.*status*(HttpStatus.***CREATED***).body(service.storeEmployee(emp));

}

@GetMapping(produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<Object> getAllApi() {

**return** ResponseEntity.*status*(HttpStatus.***OK***).body(service.fetchAllEmployees());

}

@GetMapping(path = "{eid}")

**public** ResponseEntity<Object> getEmployee(@PathVariable("eid") **int** id) {

ResponseEntity<Object> response = **null**;

**try** {

Employee employee = service.fetchEmployee(id);

response = ResponseEntity.*status*(200).body(employee);

} **catch** (EmployeeNotFoundException e) {

String err = e.getMessage();

SimpleResponse sr = **new** SimpleResponse();

sr.setMessage(err);

response = ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).body(sr);

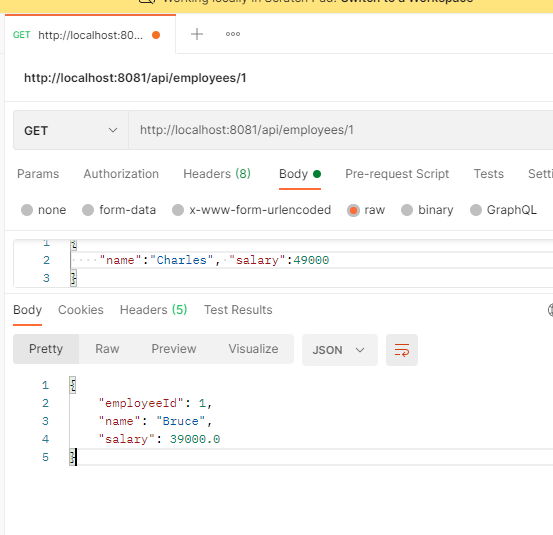
}

**return** response;

}

}

Output:



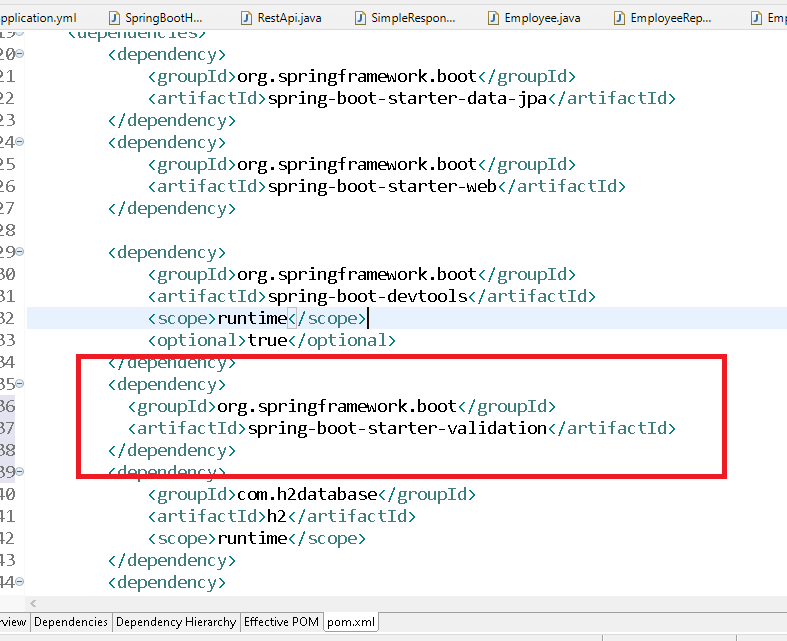
Bean Validations:

It is used to validate the data so that you can avoid invalid values, like name not to be empty, salary to be always positive, size of the characters, min value, max value and so on.

We have annotations to identify the values to follow the validation rules like @Size, @NotEmpty, @Min, @Max, @Positive, @Negative and so on.

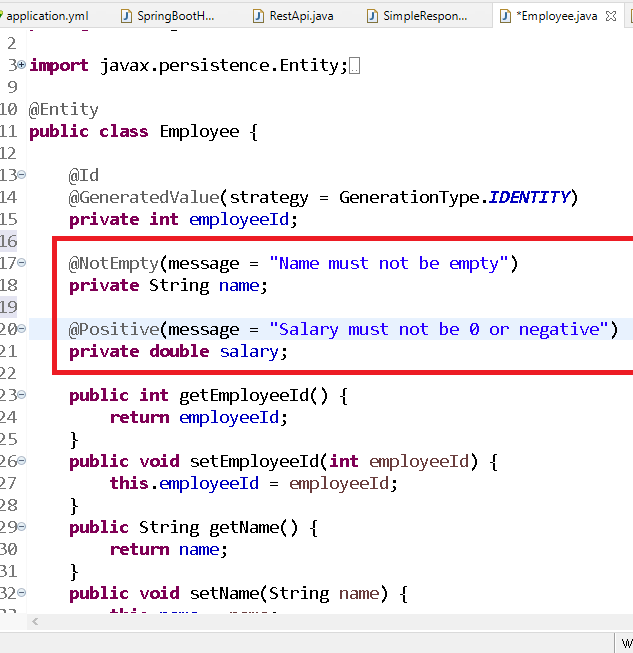
These annotations are part of spring boot starter validator and the annotations must be written on top of the bean properties.

Pom.xml



Now you can configure the beans properties to have validator annotations

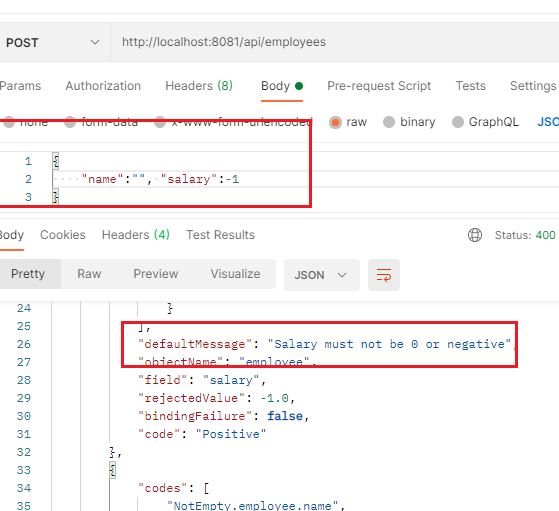
Employee.java



Now in RestAPI you have to mention that request must have valid data using @Valid annotation



Now if you pass invalid inputs for name & salary you will get an exception

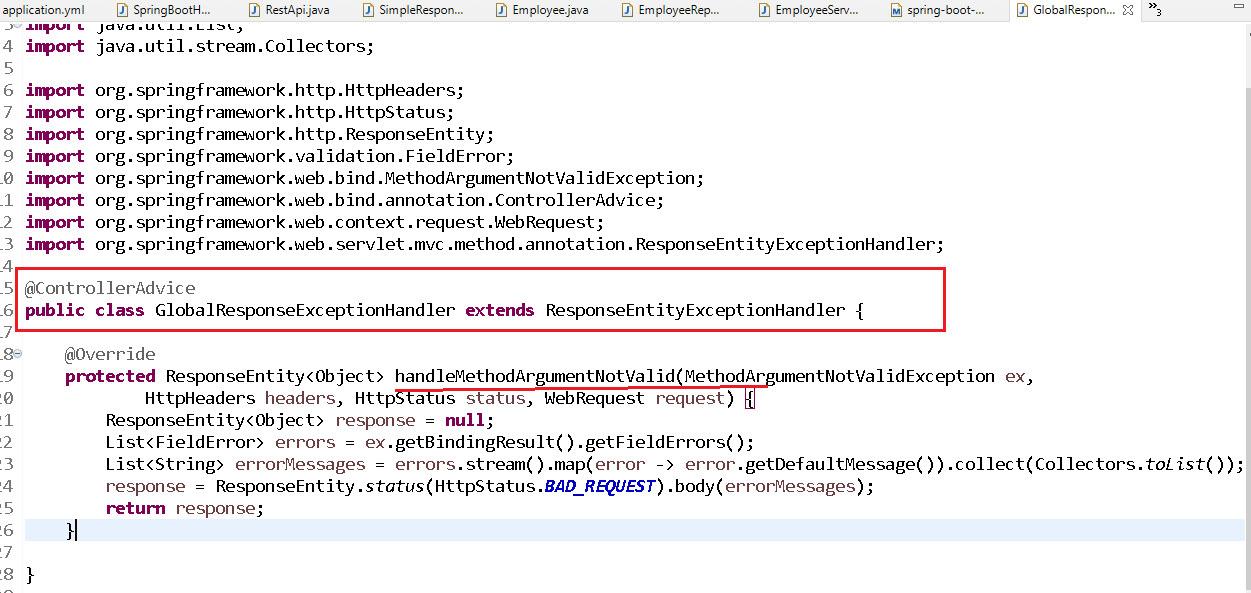


Since the response has so many informations we can customize it by handling the exceptions and create the response that is simple to understand or as per the requirement, the response what is returned is the default response when the exception is not handled, however we can handle the exceptions if in case exception occurs at the controller and provide a customized response instead of default response which will too much informations that may not be necessary all the times, this is also called as global exception handler.

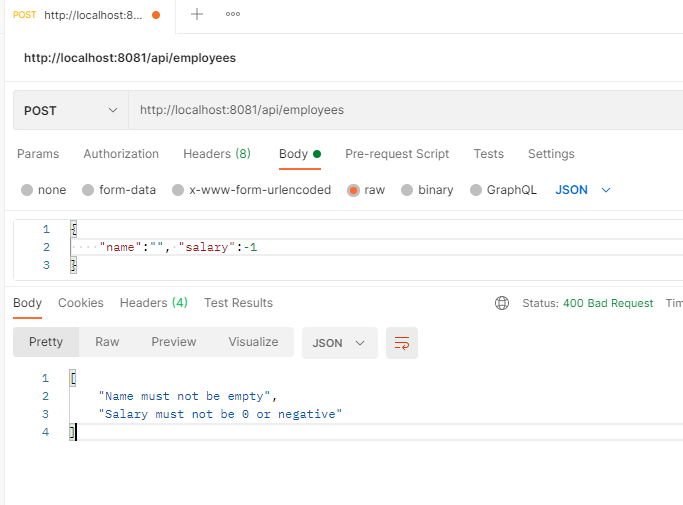
Global Exception Handler

The handler for all the exceptions occurring in the application is called as global exception handler, for that we need to use @ControllerAdvice annotation on top of the class that extends one exception class related to ResponseEntity i.e., ResponseEntityExceptionHandler

GlobalResopnseExceptionHandler.java



Output:



Spring Security with OAuth2.0

It is used to enable authorization & authentication feature for your webservice, by default the webservices accept request from all types of users, however you can enable security so that only authorized and authenticated user can access the webservices.

OAuth2.0 is the authentication & authorization framework which enables user to authenticate through tokens

There are 4 important components used in the OAuth2.0

1. Auth server: generates the token & validates the token
2. Protected Resource: which will have the webservice
3. Resource Owner: who gives permission
4. Application client: this is the application the end user uses, if the application is registered in the Auth server then only user can access the service, ex: Youtube is an application that can share videos on facebook because youtube application is registered in facebook, here the credentials of youtube is necessary as well the user credentials is also necessary, which is why when you want to share youtube videos on facebook it asks you to enter your credentials so that it will be posted on your facebook wall, however it doesn’t ask credentials of the youtube application because that is sent internally.

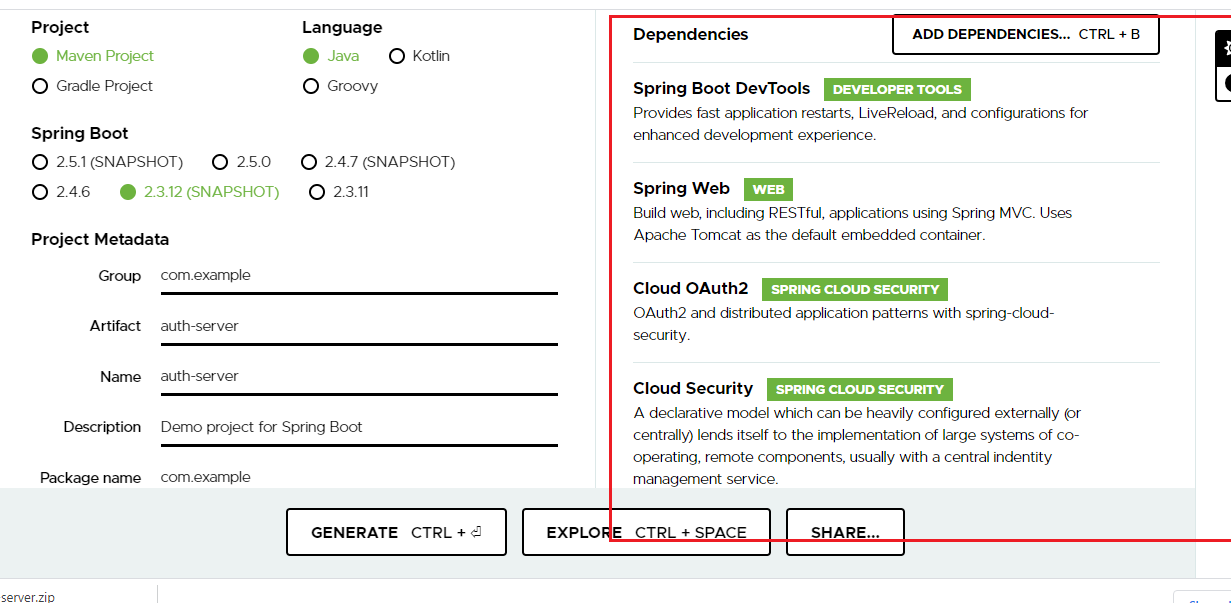
Project dependencies

1. Cloud OAuth2
2. Cloud Security
3. Spring Web
4. Dev tools

Here we need to create 2 projects

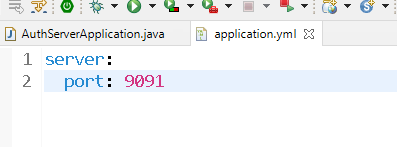
1. OAuth2.0 server
2. Protected Services

First project is to create OAuth2.0 server which can detect the user & the client application to generate & validate the token



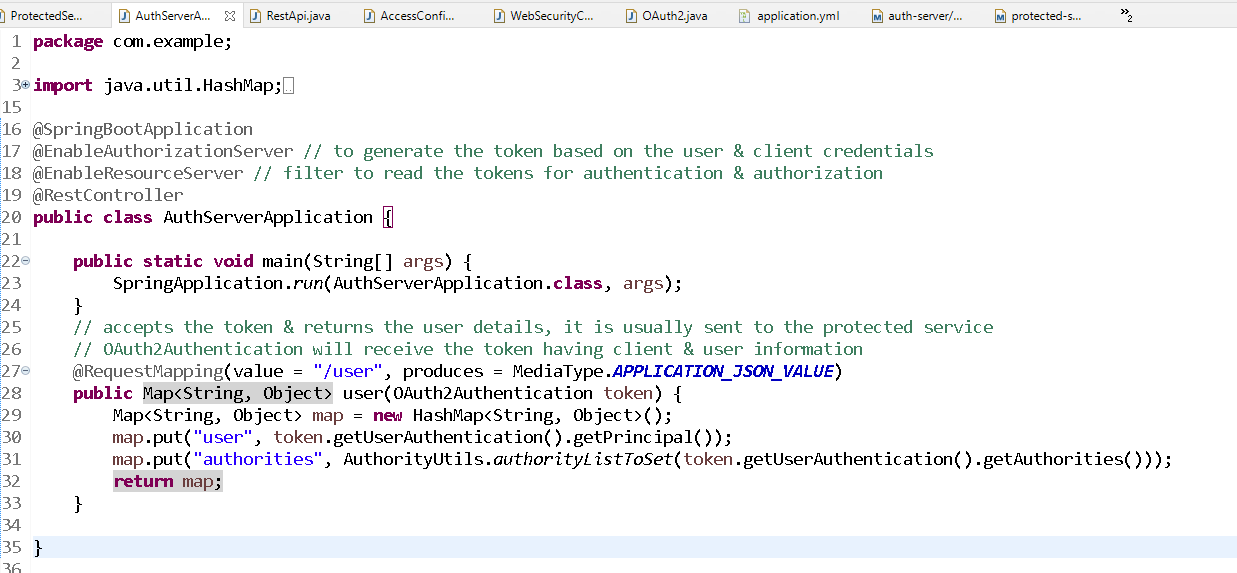
We are going to use in-memory users with roles & clients instead of checking them in the database

application.yml



We need to create an OAuthserver & make that to generate token as well read the tokens to validate & verify the user & the client, for that we need to use @EnableAuthorizationServer to create oauth server & @EnableResourceServer to intercept the token for verification & validation.

AuthServerApplication.java



OAuth2Authentication: It is in the parameter that will receive the token having client & user information, from that you can get the user & their roles.

getPrincipal(): it gives username

getAuthorities(): it gives user roles like admin, user user, guest & etc.

Now we need to configure/register the application which is used to authenticate

We have to use hard-code client details

WebSecurityConfigurer.java



The above class configures the user, authentication manager bean and user details bean

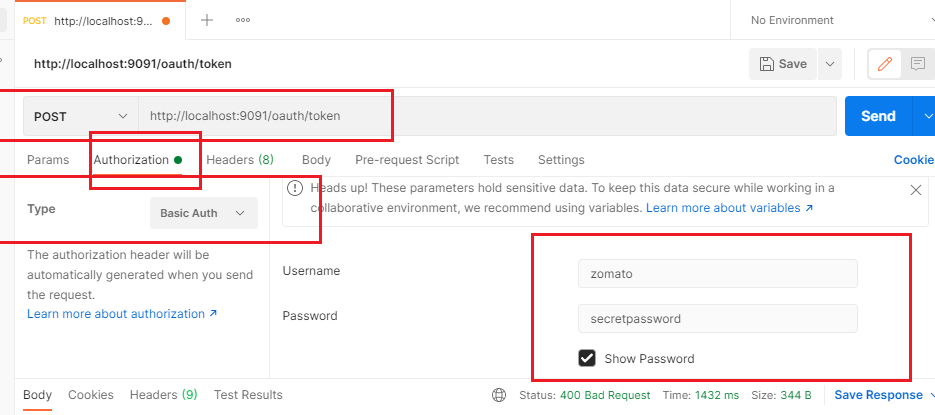
OAuth2.java



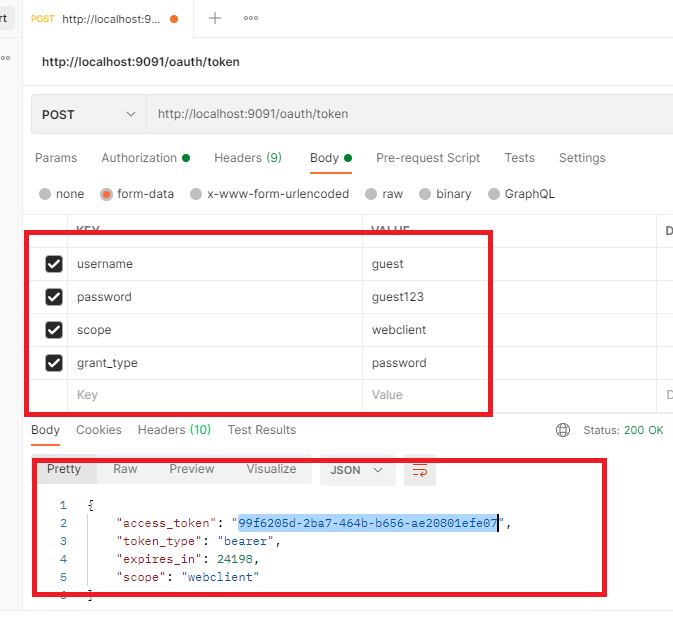
Now you can enter user credentials and also the application credentials to generate & validate the token

Setting up the postman to enter client & user information

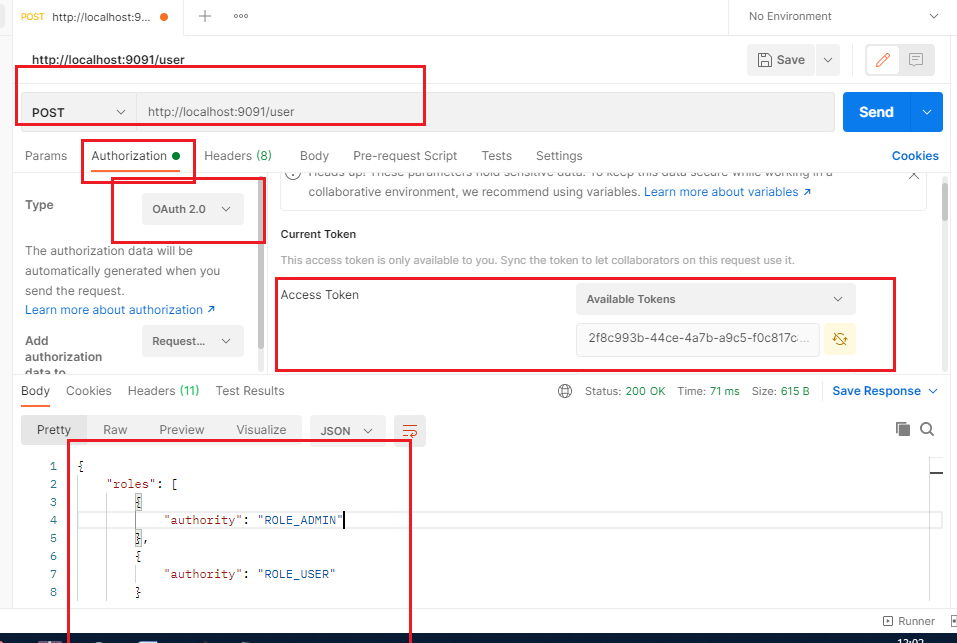
First Step: Choosing basic auth & entering client credentials



Second Step: Choose the Body & formdata & Entering the user credentials & grant-types, scopes of the client



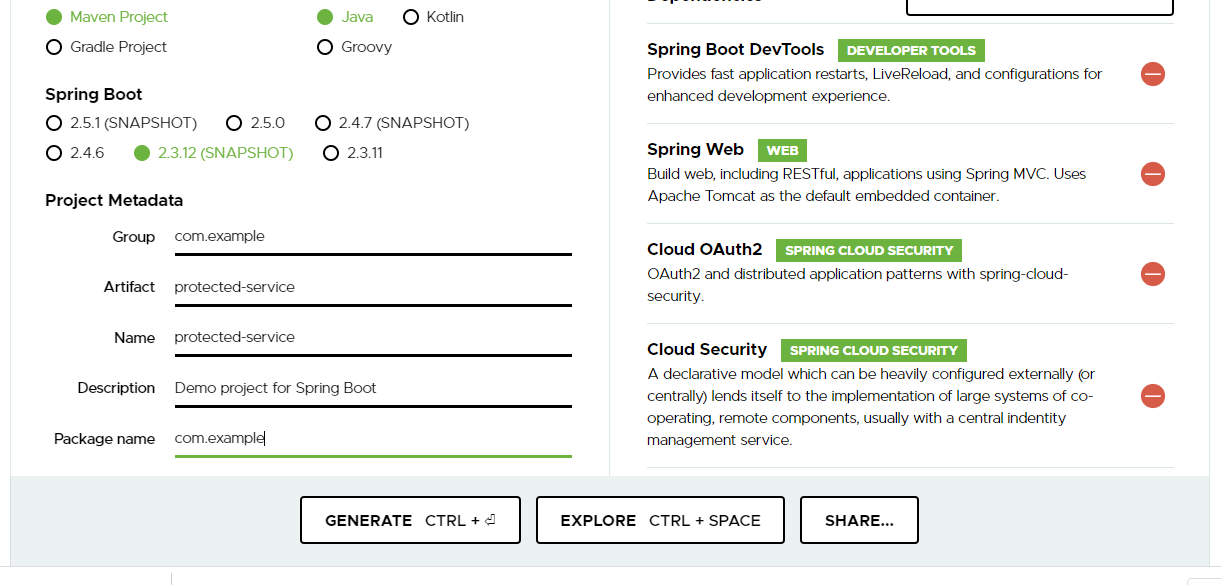
Now this token should be used to get the access to the service, if anything is wrong you will get exception



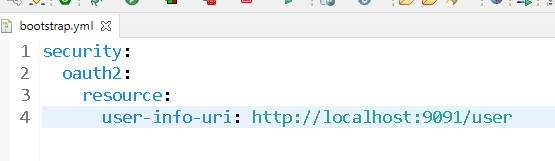
The above part is done in the protected resource which will receive the token and sends the token to the oauth server using the url <http://localhost:9091/user>

Creating the protected resources that can receive the token and authenticate using OAuth server

Dependencies



bootstrap.yml



bootstrap.yml is very first file loaded before application.yml, ideally you keep some informations that has to be loaded before application.yml loads, like connecting to some backend repository or configuring user endpoint of oauth server, application name and so on.

application.yml

server:

port: 8082

Creating a resource that can accept the token using @EnableResourceServer

ProtectedServiceApplication.java

**package** com.example;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableResourceServer

**public** **class** ProtectedServiceApplication {

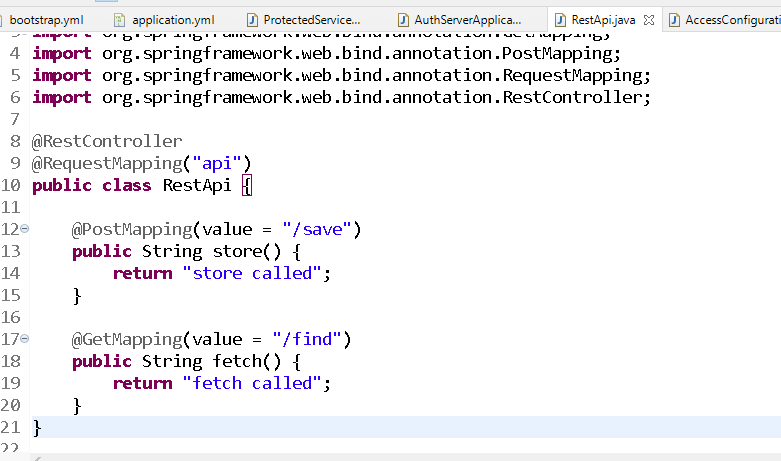
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(ProtectedServiceApplication.**class**, args);

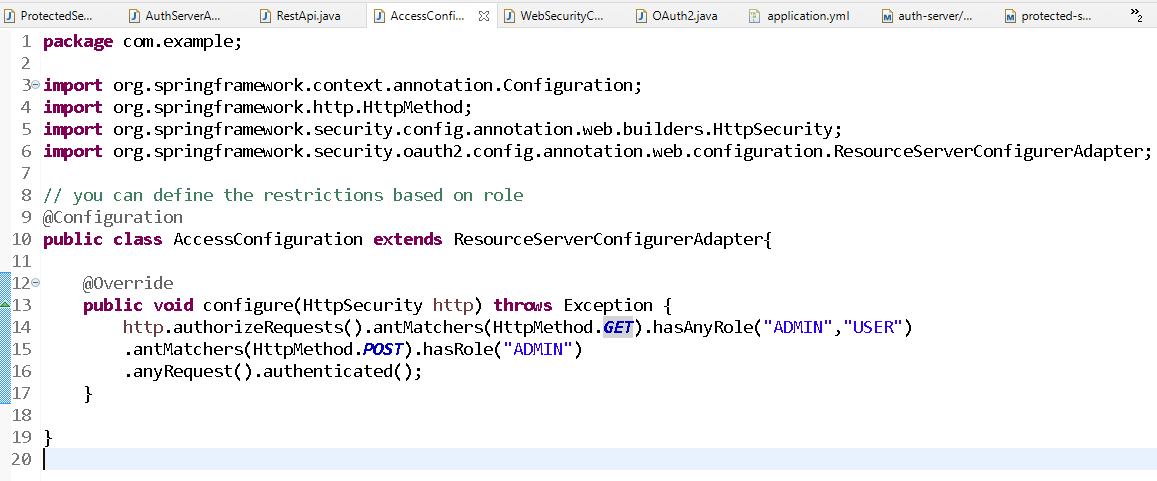
}

}

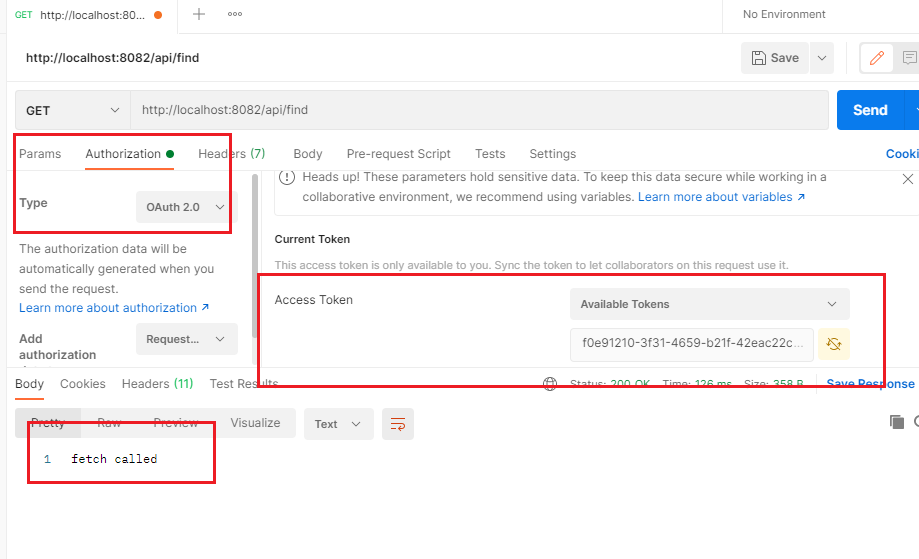
RestApi.java



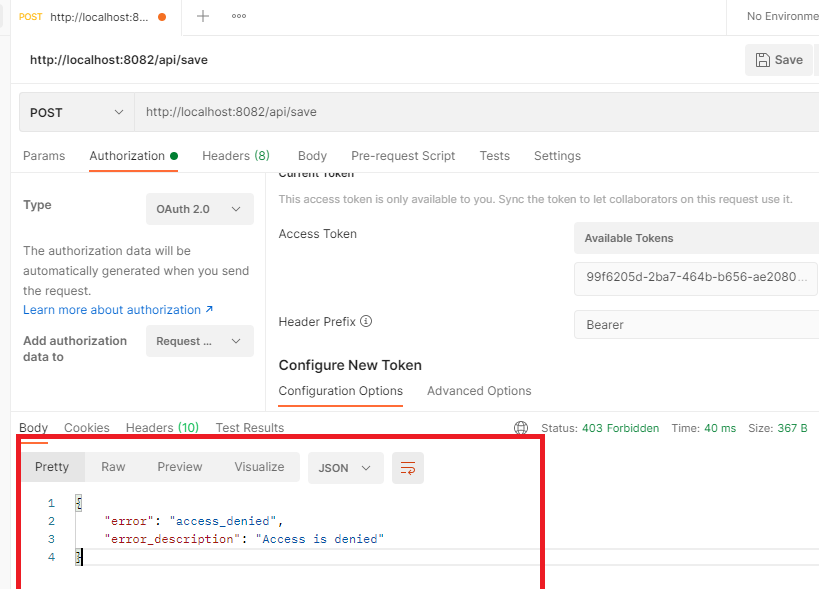
AccessConfiguration.java



Output:



Accessing the post from User role token



JWT:

OAuth2 is token based authentication framework which doesn’t provide any standard like how the token should be generated or how the tokens must be read, but JWT follows the standard

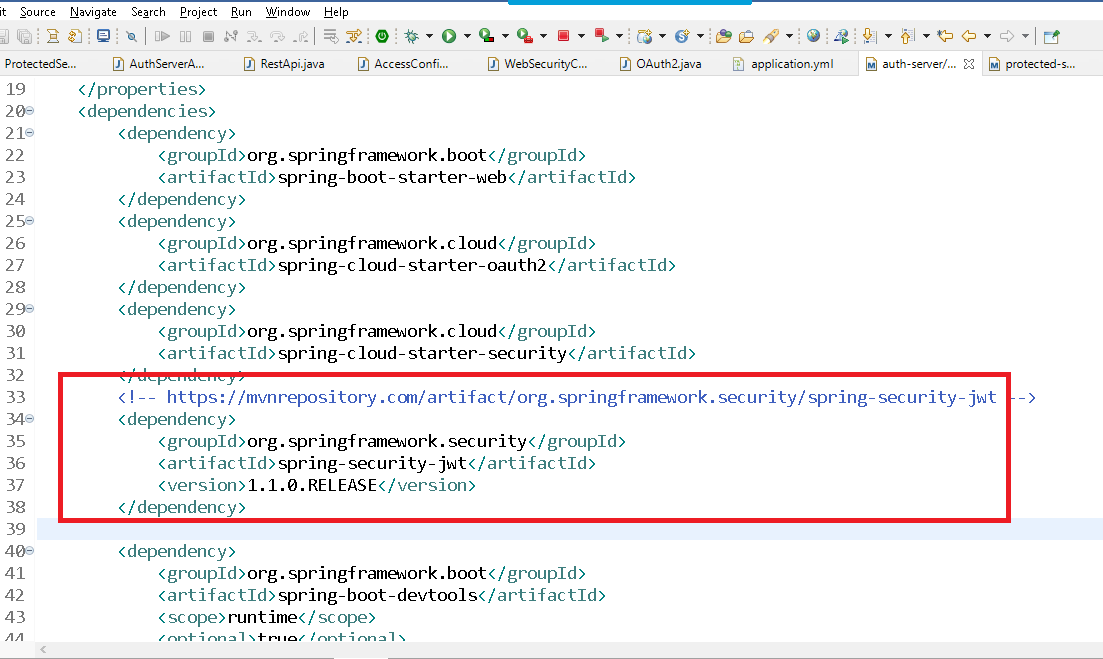
JSON Web Token: It is also used to achieve token based authentication, since OAuth2.0 tokens don’t have a standard we can use JWT for Authentication & Authorization.

The program will be slightly modified in OAuth Server to generate the JWT

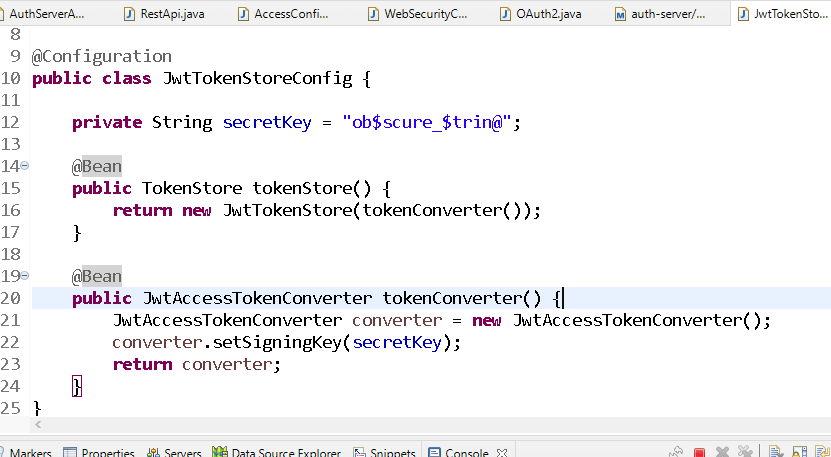
Dependencies

Spring-security-jwt

pom.xml



Creating JWT enhancers



Modifying the OAuth2.java to generate the JWT

**package** com.example;

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

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.authentication.AuthenticationManager;

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

**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.configurers.AuthorizationServerEndpointsConfigurer;

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

**import** org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;

// this is used to register the client application

@Configuration

**public** **class** OAuth2 **extends** AuthorizationServerConfigurerAdapter {

@Autowired

**private** UserDetailsService userDetails;

@Autowired

**private** AuthenticationManager authentication;

@Autowired

**private** TokenStore tokenStore;

@Autowired

**private** JwtAccessTokenConverter jwtAccessTokenConverter;

// automatically called, this registers the client i.e., application client with password, scope, grant types

@Override

**public** **void** configure(ClientDetailsServiceConfigurer clients) **throws** Exception {

clients.inMemory()

.withClient("zomato")

.secret("{noop}secretpassword")

.authorizedGrantTypes("password", "client credentials", "refresh token")

.scopes("webclient", "mobileclient");

}

// configuring authentication manager & user details

@Override

**public** **void** configure(AuthorizationServerEndpointsConfigurer endpoints) **throws** Exception {

endpoints

.tokenStore(tokenStore)

.accessTokenConverter(jwtAccessTokenConverter)

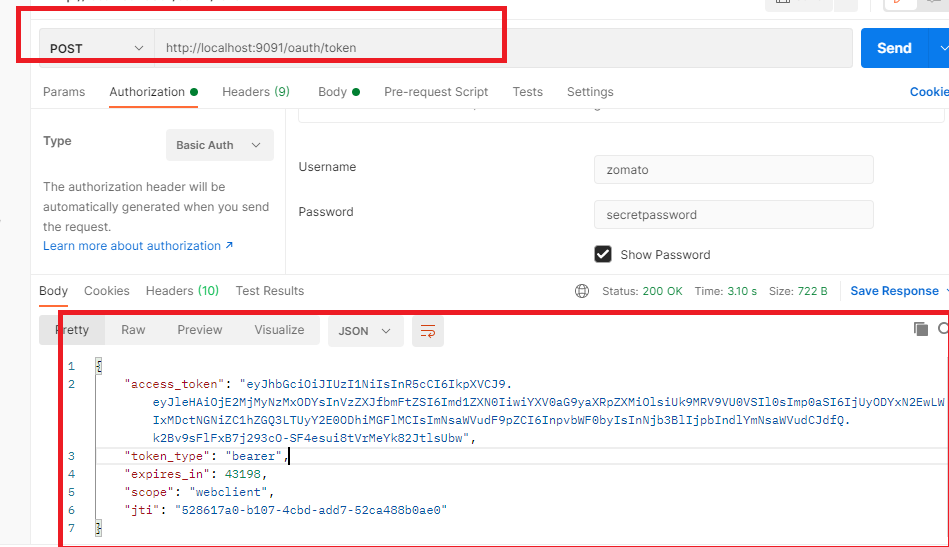
.authenticationManager(authentication)

.userDetailsService(userDetails);

}

}

Output:



The above token is for guest, which can be passed while accessing the protected service

3 – day’s project enhancement Note: Ignore Feign Client or any other microservice features

Note: Use JSP for View

Note: Has to be done by each participant’s independently

Project Name: Return Order Management