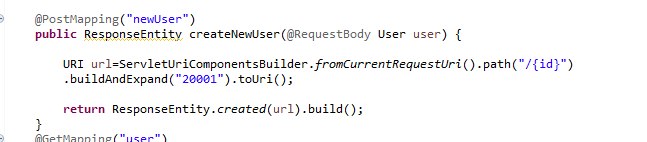
**Spring micro service**

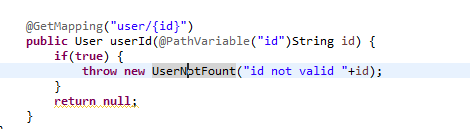
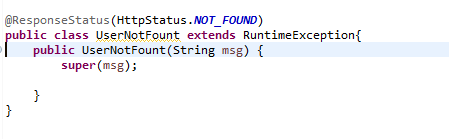
Http post respone send correct response code 201



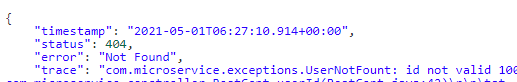
Output



Throwing exception and mapping httpstatus with those exception.

1.  2. 

Output:



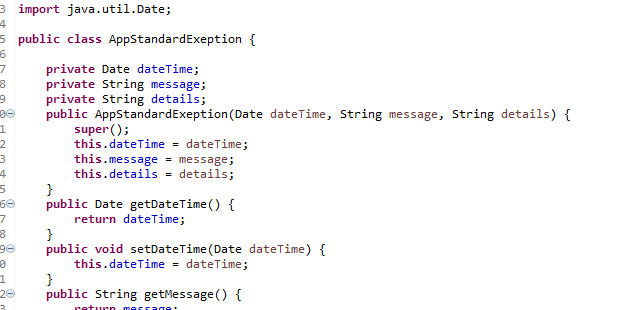


Note: if you normal throw any error without mapping response code that will display message “Internal error and response code 500”

**How to standardising Exception Handling**

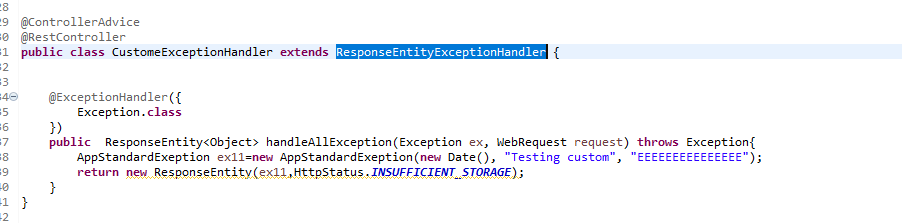
Point 1 .

Create a standard Format (class )



Step 2:

Extends ResponseEntityExceptionHandler class



Output :

Note : **One thing you must rember you need to override all method else it show only error code**

**Validation Rest api with custome error (basic validation mention in spring 5 doxc file)**

When you extend ResponseEntityExceptionHandler

You need to override below mention method of abastract class fo custome error handler

**protected** ResponseEntity<Object> handleMethodArgumentNotValid(

MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request) {

String defaultMessage=**null**;

**for**(ObjectError s:ex.getBindingResult().getAllErrors()) {

defaultMessage=s.getDefaultMessage();

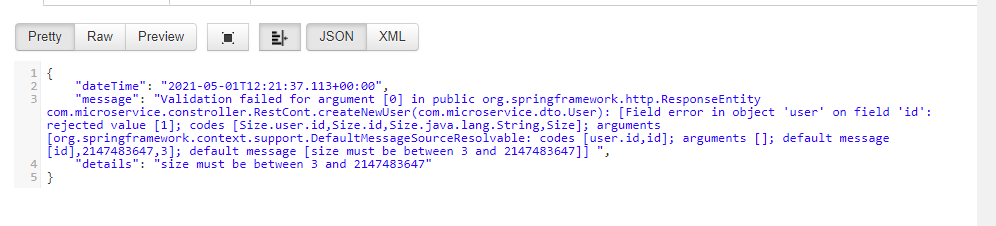
}

AppStandardExeption ex11=**new** AppStandardExeption(**new** Date(), ex.getLocalizedMessage(), defaultMessage);

**return** **new** ResponseEntity(ex11,HttpStatus.***BAD\_REQUEST***);

}

**Output**

****

**Passing link to other request in response body (hateoas)**

<dependency>

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

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

</dependency>

**import** org.springframework.hateoas.EntityModel;

**import** org.springframework.hateoas.server.mvc.WebMvcLinkBuilder;

@GetMapping("user")

**public** EntityModel user() {

User user=**new** User();

user.setDob(**new** Date());

user.setId("1");

user.setName("Deepak");

EntityModel<User> em=EntityModel.*of*(user);

WebMvcLinkBuilder builder=

WebMvcLinkBuilder.*linkTo*(WebMvcLinkBuilder.

*methodOn*(**this**.getClass(), "1").userId("1"));

em.add(builder.withRel("with param"));

**return** em;

}

output



**Internationalize service**

**import** java.util.Locale;

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

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

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

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

**import** org.springframework.context.support.ResourceBundleMessageSource;

**import** org.springframework.web.servlet.LocaleResolver;

**import** org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver;

**import** org.springframework.web.servlet.i18n.SessionLocaleResolver;

**import** org.springframework.context.MessageSource;

**import** org.springframework.context.i18n.LocaleContextHolder;

Step 1 : create default locale

@Bean

**public** LocaleResolver loaclResolver() {

AcceptHeaderLocaleResolver localeRes=**new** AcceptHeaderLocaleResolver();

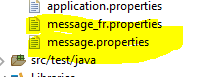
localeRes.setDefaultLocale(Locale.***US***);

**return** localeRes;

}

Step 2 : create properties file with baseName “message.properties” and other properties “message\_fr.properties”

In these two properties baseName is “message”



Note : key inside properties file should be same

Step 3 : ResourceLoader Bean

@Bean

@Primary

**public** ResourceBundleMessageSource bundleMessageSource() {

ResourceBundleMessageSource resourceBundleMessageSourceObj=**new** ResourceBundleMessageSource();

resourceBundleMessageSourceObj.setBasename("message");

**return** resourceBundleMessageSourceObj;

}

Step 4 :

Autowired messageSource and getting message properties file base on header “Accept-Language”

@Autowired

**private** MessageSource messageSourceObj;

@GetMapping("hello-world-inter")

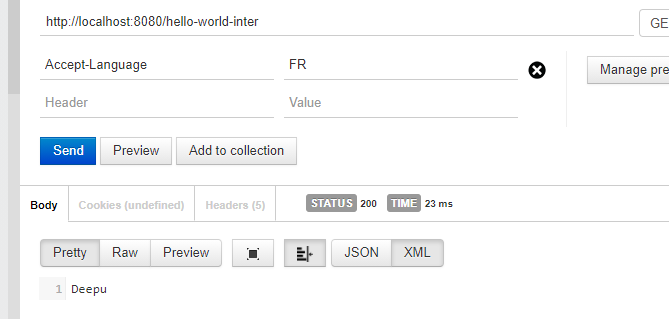
**public** String helloWorldInter() {

System.***out***.println(LocaleContextHolder.*getLocale*());

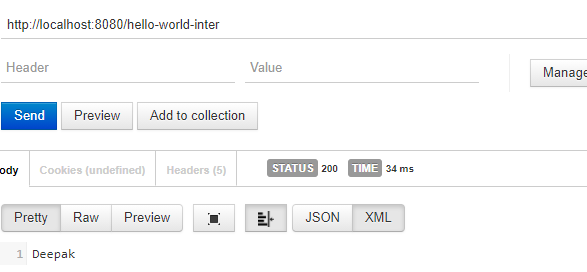
**return** messageSourceObj.getMessage("ms", **null**, LocaleContextHolder.*getLocale*());

}

Output : 1



Output : 2



**Enable XML response From api**

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml -->

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

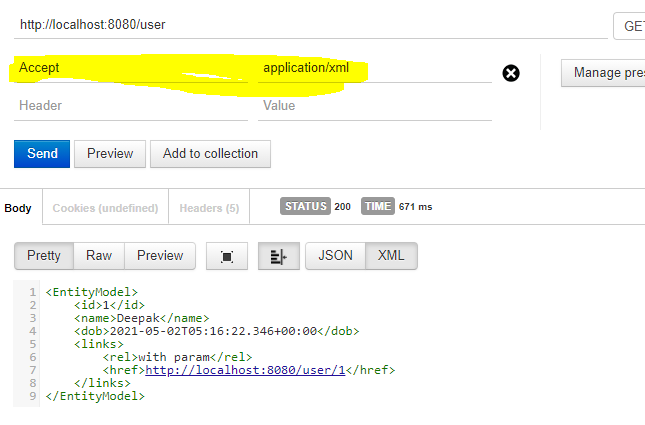
<artifactId>jackson-dataformat-xml</artifactId>

<!-- <version>2.12.1</version> -->

</dependency>

**Restart server**

**Output :**

****

**Implemention sweger dumentation**

Step 1:

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-boot-starter</artifactId>

<version>3.0.0</version>

</dependency>

Step 2:

Creating sweger configuration class

**package** com.microservice.constroller;

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

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

**import** springfox.documentation.spi.DocumentationType;

**import** springfox.documentation.spring.web.plugins.Docket;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration

@EnableSwagger2

**public** **class** SwegerConfig {

@Bean

**public** Docket api() {

**return** **new** Docket(DocumentationType.***SWAGGER\_2***);

}

}

Step 3 :

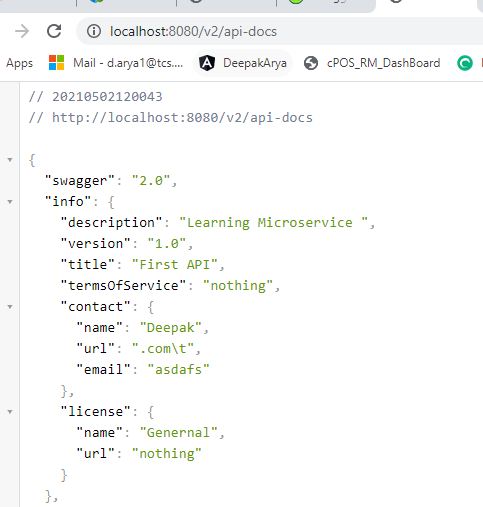
URL : <http://localhost:8080/swagger-ui/>

URL 2 : <http://localhost:8080/v2/api-docs>

**Changing information is sweger documention:**

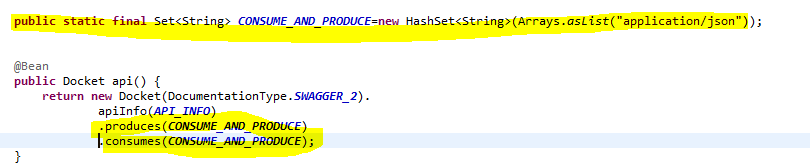
****

**Output :**

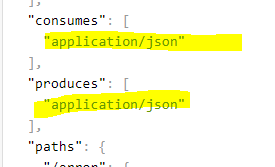
****

**Changing information is sweger documention:**

****

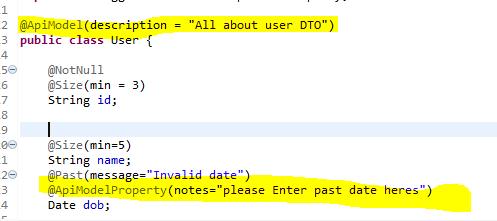
****

Output:



**Adding Discriptin to data model or DTO (for documentiation)**

****

****

Output



**Some other annotation**

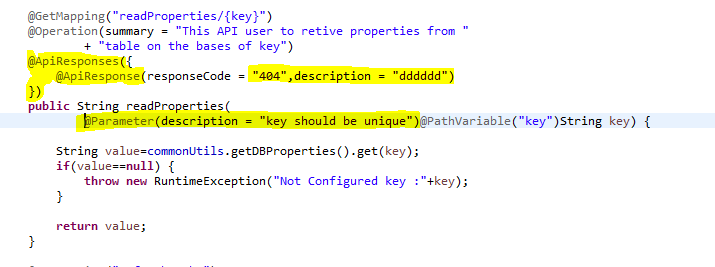
**import** io.swagger.v3.oas.annotations.Operation;

**import** io.swagger.v3.oas.annotations.Parameter;

**import** io.swagger.v3.oas.annotations.responses.ApiResponse;

**import** io.swagger.v3.oas.annotations.responses.ApiResponses;

@Operation(summary = "To access user details you can use this api") # for providing description about api

****

**\*Note : make sure all annotation must import from sweger version 3**

**Use spring-data-rest-hal-explorer and acutor**

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-rest-hal-explorer</artifactId>

</dependency>

Application properties :

management.endpoints.web.exposure.include=\*

**urls:**

[**http://localhost:8080/**](http://localhost:8080/) **# hel browser**

<http://localhost:8080/actuator> # actuator

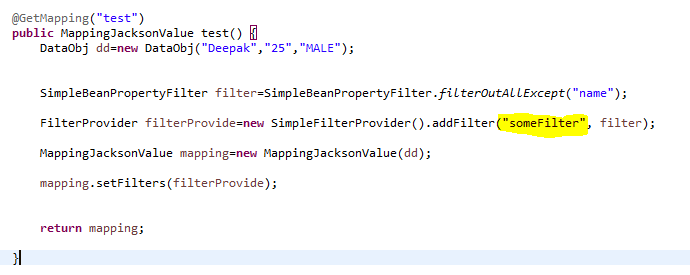
How to static filter out values from response

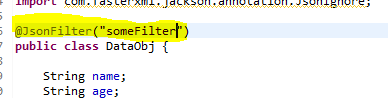
Ans : by using @JsonIgnore annotation on properties

What ever properties having @JsonIgnore will not display in response

**Dynamic filtering :**

****

****

****

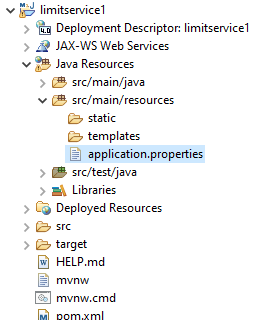
**Versioning :/ params /header/producer**

****

****

**Configuring centerlize server :**

**Step 1 create new application Limit server**

****

**Step 1.1**

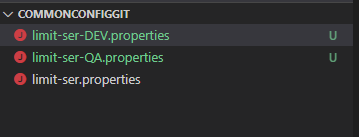
Create a simple rest service which is reading data from application.properties and display that



**Step 2 create a spring boot could sever config application**

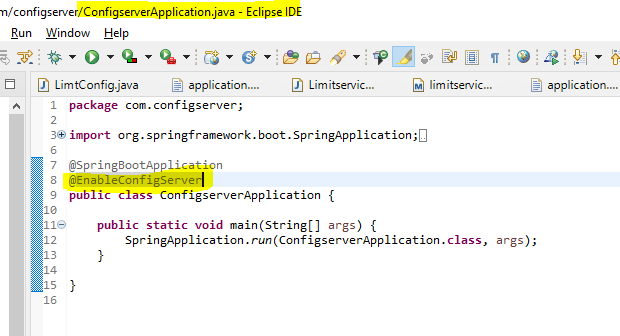
**** ****

**Step 3 . create local git repository (git init) and add configuration properties:**

****

**Namming convention of file -> applicationName-ENV.properties**

**Step 4 : Enable server configuration:**

****

Note : this annotation on server application main file

**Step 5 : let discuss little about limit service config application poperties**

server.port=8080

spring.application.name=limit-ser -----🡪application name

spring.config.import=optional:configserver:http://localhost:8888 --🡪 address config server

spring.cloud.config.profile=DEV --------------🡪Enviremnt

limit.min=1

limit.max=1000

\*note : as we keep properties file on config server with applicationName-ENV ---- for this application limit-ser-DEV.properties file will load.

**Step 6 : let dicuss little about config server properties**

spring.application.name=config-server -🡪application name

server.port=8888

spring.cloud.config.server.git.uri=file:///E:/Learning/Projects/Archived/commonConfigGit -🡪 git config folder path

Note :

* If you have not implement server side yet and you are using client dependacy in client

project.your application is not going to start …

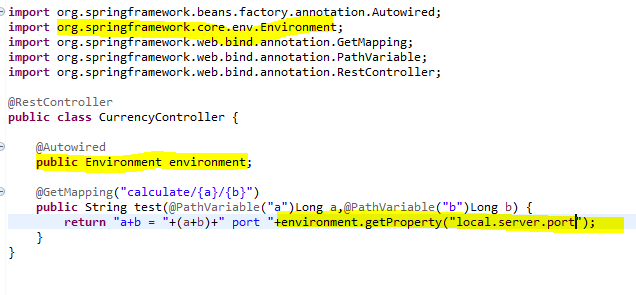
please set below mention property to start server

spring.cloud.config.import-check.enabled=false

URL:

#http://localhost:8888/<context-root>/<application-name>/<profile>

**How to get port where your service is running**

****

**Sending Get and post Request using RestTemplate :**





**How to pass Header in http get request**

****

Rest api call using **Feign**

**Step 1**

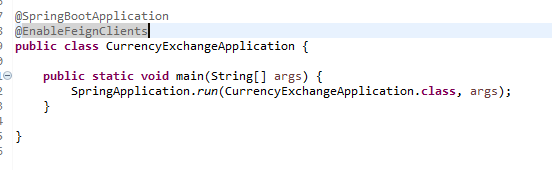
<dependency>

<groupId>org.springframework.cloud</groupId>

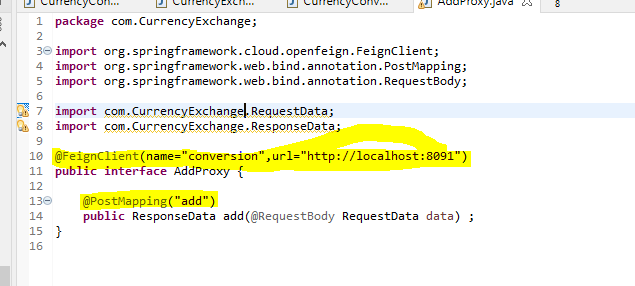
<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

Step 2. Enable Feign Clients



Step 3.



Step 4

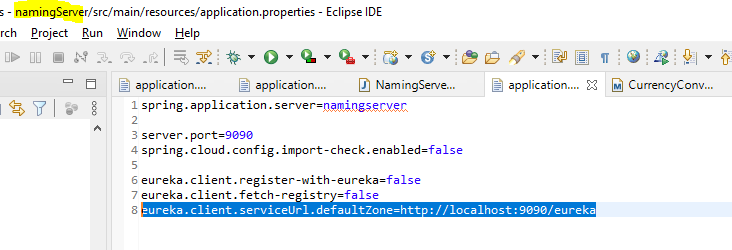


**Euraka Namming server**

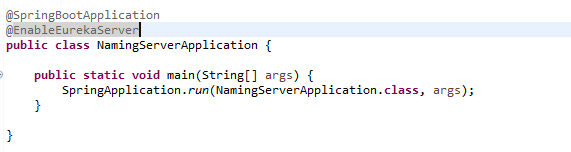
**Step 1: Create new server**

****

**Step 2:Namming server application properties file.**

****

**Step 3:enable eureka server**

****

**Step 4 : Register client under (Client project)**

**4.1 add dependancy**

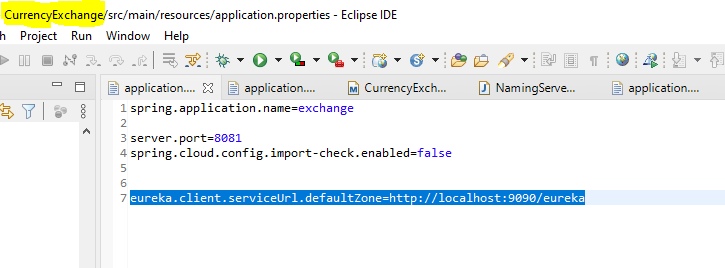
<dependency>

<groupId>org.springframework.cloud</groupId>

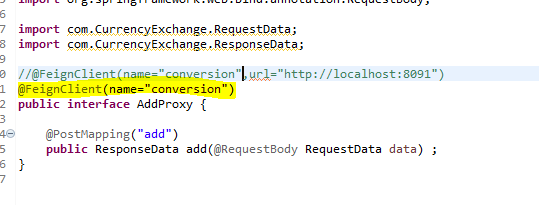
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

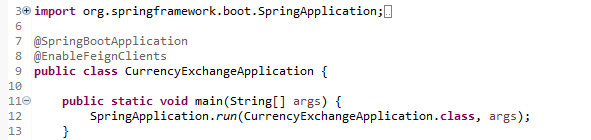
**4.2 add URL in client application.properties**

****

**Step 5.Change Feign proxy class to Hit by Load balancer**

****

**Step 6 .enable feign**

****

**Implementing API Gate way**

**Step 1 . create project.**

****

**Step 2:**

**Add below mention properties to all the micro service application.properties**

eureka.instance.prefer-ip-address=true

step 3:

Old url :

<http://localhost:8081/calculateF/1/2>

API gate way Url:

<http://localhost:1234/EXCHANGE/calculate/1/2>

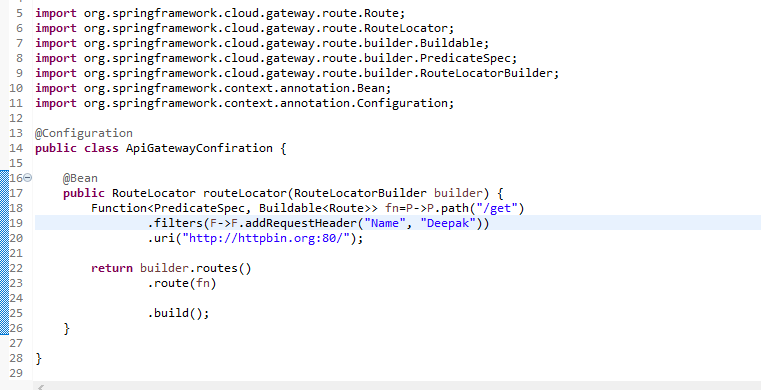
Note: /EXCHANGE/ -- > application name in name server

**If you want to use application name in url in lowercase then set below mention properties in api gateway application.properties**

spring.cloud.gateway.discovery.locator.lowerCaseServiceId=true

**API Gateway adding param and header to request when it pass throw API Gateway**

****

****

Line no 19 : adding header to request every request for “URL : /get”

Line no 20 : redirect each request “get”-> ”http://httpbin.org:80/”

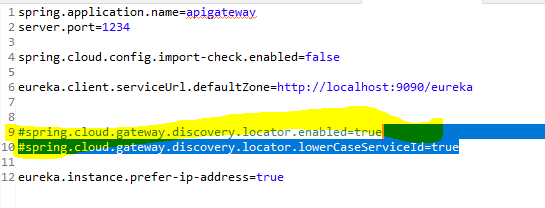
API gateway simply URL calling of micro service

Example :

Currently with gatewayAPI -> <http://localhost:1234/exchange/calculate/1/2>

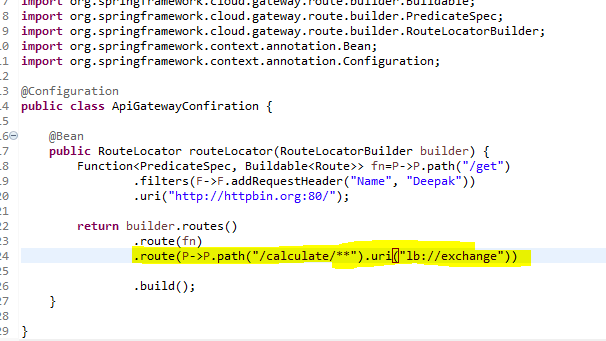
\*new Changing this to -🡪 <http://localhost:1234/calculate/1/2>

Step 1 . comment API gate way properties which are automatically managing change.

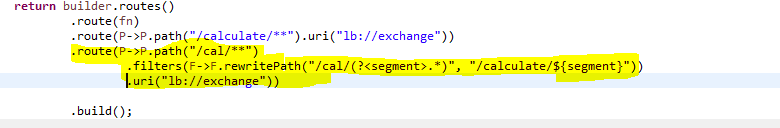


Step 2: configure route manually





**How to rewrite path**

****

**API Gateway global Filter**

****

****

**Foalt tolrence and regligin (retry, cirecutBreaker, rateLimit)**

**Step 1: create project**

****

**Step 2: create controller**

****

**Step 3: Application.properties**

****