Rest Api

**Created by**

**Saurav Kumar Jha(**[**321sauravkumar@gmail.com**](mailto:321sauravkumar@gmail.com)**)**

**Note:** you can’t sell or re-distribute without the consent of the author

**REST**

transfer(send/receive)

state

representational

🡪what represent state in object oriented language

object represent the state and fields holds the state in object

🡪we have learned this in transaction like new state,stable etc etc,

methods don’t hold the state

suppose we have railway monolithic app we are sending the request and giving the html as response

indian railway

user

frontend

backend

request

response

processing happens here

DB

this is fine but instead the book ticket directly from railway app we can also book the railway ticket using makeMyTrip and paytm and other application in that case

request1 Indian railway server

MakeMyTripp

user1

java backend

dao layer methods

service layer methods

request2 traindetails

response response

ticketdetails

DB

bookTicket

Paytm

endpoints

user2

makeMyTrip has only view not the model(data) so when user will request to the makeMyTrip let’s say request1 then makeMyTrip will internally request to railway server request2 for the data then it will get the response from railway server then it will give back the response to the user same for paytm

now problem is that you are saying that railway server will give the object to the paytm as the response and railway backend is written in java and now paytm let’s say javascript application or php application etc and paytm application will no bet having jvm and railway server is sending java object means bytecode and paytm

application doesn’t have jvm then how it will use the object so railway server will not send the object ,it will send the representation of object and obviously this representation should be language independent, platform independent etc like xml or json

and in spring there is a default support for json

now railway server has java object so it will not send the java object it will send its representation that is json so now if paytm side is either php application or ruby application etc it will process the json and same for request data just like response data

java server

railway server

response

request

representation

{“id”:1,”name”:”gatimaan”}

request request(json)

user

response response(json) java

php/ruby/python/java etc

request request(json)

response response(json)

user frontend(Angular) java server

now in REST we work on resources , we do CRUD operation on resources , resource can be Product, Author, Book etc

C🡪create (POST request method)

R🡪read/fetch (GET request method)

U🡪change (PUT request method but we can also use POST but conventionally wrong)

D🡪delete (DELETE request method)

REST is based on the http protocol

HTTP response

HTTP request

Header

Body(data)

Header

Body(data)

content-type

status code etc

in GET request we will send the data as the part of url

for ex🡪 <http://localhost:8585/trains/id=2>

here id is path variable(@PathVariable)

or

<http://localhost:8585/trains?id=2>

here id is request parameter(@RequestParam)

but in REST path variable is most popular

in POST method generally we will send the data as the part of the body not url it means you will send the lot of data And it makes sense, also because, then you are creating a resource, you have to send a lot of data that data is not possible, you cannot send in url

For updating(PUT method) again your have to send a lot of data, data you can send as the part of body , same with DELETE

Q. what is @RestController

🡪 @RestController annotation is a special controller used in RESTful Web services, and it's the combination of @Controller and @ResponseBody annotation automatically so we do not have to add @ResponseBody to our mapping methods.. It is a specialized version of @Component annotation

🡪 RestController is used for making restful web services with the help of the @RestController annotation. This annotation is used at the class level and allows the class to handle the requests made by the client

Q. difference between @Controller and @RestController

| @Controller | @RestController |
| --- | --- |
| @Controller is used to mark classes as Spring MVC Controller. | @RestController annotation is a special controller used in RESTful Web services, and it’s the combination of @Controller and @ResponseBody annotation. |
| It is a specialized version of @Component annotation. | It is a specialized version of @Controller annotation. |
| In @Controller, we can return a view in Spring Web MVC. | In @RestController, we can not return a view. |
| @Controller annotation indicates that the class is a “controller” like a web controller. | @RestController annotation indicates that class is a controller where @RequestMapping methods assume @ResponseBody semantics by default. |
| In @Controller, we need to use @ResponseBody on every handler method. | In @RestController, we don’t need to use @ResponseBody on every handler method. |
| It was added to Spring 2.5 version. | It was added to Spring 4.0 version. |

🡪Spring converts the object to json format automatically, If spring was not doing that, then it becomes your responsibility to convert but spring is converting.

🡪what is @RequestBody

@RequestBody annotation maps the HttpRequest body to a transfer or domain object, enabling automatic deserialization of the inbound HttpRequest body onto a Java object. Spring automatically deserializes the JSON into a Java type, assuming an appropriate one is specified

using @RequestBody we are mentioning that json data which is there in the body of request convert to object,spring will intercept and convert from json to object if spring is not doing that you will have to do by yourself

🡪what is @RequestMapping

@RequestMapping annotation is used to map web requests onto specific handler classes and/or handler methods. @RequestMapping can be applied to the controller class as well as methods.

🡪All the mapping like POST, GET, PUT,DELETE is the shortcut of @RequestMapping only

suppose we have @GetMapping(“/products/{id}”)

so if we write in the form of @RequestMapping then

@RequestMapping((“/products/{id}”,method = RequestMethod.GET)

same with others

🡪difference between PUT and PATCH

If you have a lot of things to change in object then PUT is used conventionally they should be used and if you want to change only one property let’s say name and or a price or something so conventionally PATCH is considered better

Till now we have not handled the exception , now we will handle exception

before that we will discuss status code and it’s very important to return correct status code

2xx – request processing is successful

for ex🡪

200:OK(default status code)

201: CREATED

202: ACCEPTED

Http error code(request processing not successful)

4xx—client side error codes

400—BAD REQUEST(validation error by the client )

401—UNAUTHORIZED

402—payment required

403—forbidden

404—not found

405—bad method

5xx—this because of server side issues

500: internal server error

503:service is not available

so now in rest Api controller class instead of throwing the exception we will handle it by using try catch block inside the particular method but by handling here we are polluting the code so we will create separate method ,on top of that method we will use @ExceptionHandler(ExceptionClassName.class) we will write it mean that method will handle these exception when it will be occurred

we can also mention the status this method will return by using

@ResponseStatus

Now, if you will not mention the correct status code then by default it will be 200 but that is wrong because there is an exception so we will mention correct status code

for example if we want to show the status 404 then we can mention like this

@ResponseStatus(HttpStatus.NOT\_FOUND)

now here need more improvement since that particular exception or similar types of exception can have at other place or in some other controller also then you have to write the same thing So instead of duplicating the code we will make the another class for example CentralizedExceptionHandler custom class inside the controller layer and inside it we will create the exception handler method

on top of the class we will mention @RestControllerAdvice

using @RestControllerAdvice we are mentioning that this class will act as a global exception handler class

@RestControllerAdvice is a special type of @Component but it also says that in this class there are methods which are handling exceptions and returning the response body

now if we have to handle two exceptions let’s say InvalidIdException and InvalidNameException then we can create separate method and top of that we can write

@ExceptionHandler(InvalidIdException.class) and @ExceptionHandler(InvalidNameException.class)

but if these two methods doing the same thing then instead of creating the separate method we can create single method and top of that we can mention

@ExceptionHandler({

InvalidIdException.class,

InvalidNameException.class

})

now we will discuss about declarative validation that is provided in java enterprise edition, there are java enterprise annotation for validation

spring also has transitive dependency on that annotations other than this it also provides little of its own validations also annotations also

for this first thing we will do that is adding spring-boot-starter-validation dependency in pom.xml file , it has transitive dependency on javax validation

now in the class where you want to add the validation , we will write @Validated on top of the class

now we use lot of annotations on fields ,parameters and methods for ex🡪

@Min(1) 🡪 it means length can’t be less than 1

@Max(23)

both @Min and @Max work with numbers only

now for handle the exception we will write @ExceptionHandler(ConstraintViolationException.class) on top of the exception handler method

now if we write @Min(1) and handle it then it will show predefind message but if we want to show our custom message then we can write @Min(value=1,message=”custom message”)

we can use @Valid annotation on request body data

we will use more validations later

**DTO(Data Transfer Object)**

DTO is a simple object , the job of this is to send data from one layer to another layer

now understand why we are using DTO

suppose we have entity class Product that contains lot of fields like id,name,price,Category object,manufacturedDate etc so we are sending this complete object as a request body and in response body also we are getting this complete object but let’s suppose user only send name as request body and in response only price and name is required so why we are sending and receiving the complete product object that will unnecessarily put load on the server so instead of this we will create simple entity dto objects , for request body we will create class inside dto package with field(name) only and for response body we will create another class with field( name and price)

so we will create class for example request body is AddProductRequest and for response body ProductDetails so in rest controller method we will create the object of ProductDetails and set all the field with the help of entity class(Product) but And we have also discussed that the rest methods should be simple, so instead of writing it here, I will create a separate class (for ex->ProductUtil) inside the util package and the conversion of product to product details etc will do here

now we can add the validations also on the field on DTO classes like

for example if we want to add the validation on the fields of AddProductRequest class

@Length(min = 1,max = 28) , @Min(2) etc

so now if we are sending this as the Request body in the controller method then this validation will only be executed if we mention @Valid in the parameter because your validations are in the separate class DTO

so for handling the exception we will add

@ExceptionHandler(MethodArgumentNotValidException.class)

now we will do more incremental development

Now we have understood that all the logic business logic ,validations complicated all the things should be there in the service layer so we have discussed before that the service layer contains the transaction management validations and all the complex business logic, so now we will improve our service layer so now instead of doing complex thing in controller we will do in service layer