Dec 01 Intro to Distributed Apps

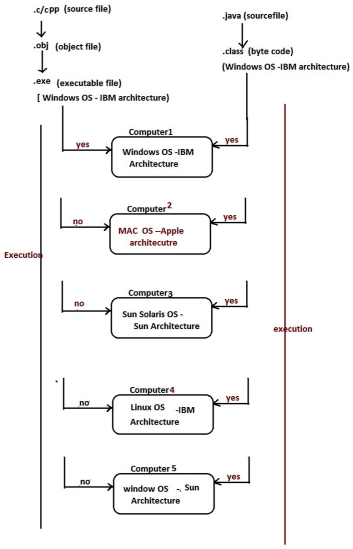
Java is platform indepenndnt and Architecture neutral

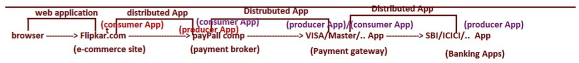
=> The plan of manifacturing vehicle is called vehicle architecture
like the four wheeler architectures are
a) Car architecture
b) Jeep architecture
c) Van architecture
d) Mini Bus Architecture
e) Mini Lorry archicture
and etc..

=> The plan /process of manifacturing computer is called computer archicture and the popular architectures are

=> IBM Architecture (all our regular laptops and desktops fall under this archicture)
=> Apple/MAC architecture
=> Sun Architecture
and etc..

=> c,c++ languages are not only Platform dependent (OS) and they are also archicture dependent where java is platform(OS) and archicture independent



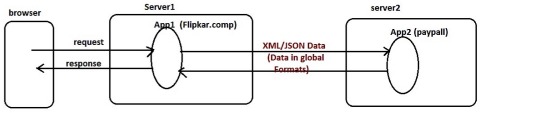


=>VISA/Master/... and etc.. Payment gateway apps providing world wide infrastructure to perform cards based (debit card/credit card) based Transactions

paypal /pay u money / rupey /razor pay and etc.. are payment broker acting as bridge b/w e-commerce Apps and payment gateways

=> The cards issued by payment gateway will come to customer through Banks by linking with bank Accounts

=>Distributed Computing / App developnent makes two Apps of two different servers belonging to same machine or different machines talking each other and also exchanging the data



=>Generally every Application we develop contains 4 layers

a) presesentation layer /UI Layer —> contains UI logics (struts ,jsf, spring mvc, servlet,jsp, angular, react js,html,js,css) (view and controller)

Model layer

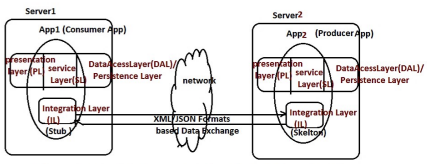
b) Serivce Layer ----------> contains b.logics (calculations, analyzations, filterings, sortings and etc..) [using java class or spring beans(spring core+spring Tx)]

c) Persinstence Layer/DataAccess Layer ------> contains persistence logic like performing CURD Operations [using jdbc, hibernate, spring jdbc, spring data jpa, spring data jdbc, spring data mongodb,spring orm and etc..]

d) Data Layer ----------> contains the data of the Application (The real persistence store) [DB s/w (SQL or No SQL) , Files (Text files, properties files, Xml files,JSON Files,...)]

if want to link one App with another App using Distributed Computing or App development env.. we need to anddtional in both cosumer and producer App that is "Integration Layer" we can that logics of this Integration layer using Distributed TEchnologies or frameworks like RMI,EJB, webservices and etc...

Consumer App —> Client App that cosumes the services of server App/Producre App
Producer App —> Server App that developes the services of server App/Producre App and keeps them ready for consumption



=> The integration layer logic of consumer App is technically called as Stub logics
=> The integration layer logic of Producer App is technically called as Skelton logics

developing
Different ways of integration logics in Java env.. /Different Distributted Technologies of Java

(a) RMI (Remote Method Invocation) | (java based)
(b) EJB (Enterprise Java Beans)
(d) CORBA (can be implemented in multiple languages)
(c) Http Invoker (from spring JEE module) | (java based)
(e) WebServices (can be implemented in multiple languages)
and etc..

other domain distributed Technologies

a) RPC (using c/c++)
b) DCOM (using micro soft technologies)
c) .Net Remoting (using .net technologies)
d) Corba (can be implemeted in muliple languages)
e) webServices (can be implemented in multiple languages)

RPC :: Remote Procedure Calls
CORBA: Common Object Request Broker Archicture
DCOM :: Distributed Computing

RMI

=>Given by Sun Ms
=> part of Jdk s/w (JSE module)
=>It is language dependent i.e both Consumer and Server App must be there in java
=> It is platform independent and Architeture neautral
=> Can not use Internet network as the communication channel b/w consumer and producer Apps\
=> Uses JRMP (Java Remote Method Procotol) as the protocol for communication
=>Outdated becoz of EJB
=> Does not give any built-in middleware services (The ready made secondary logics like security, Transaction mgmt and etc..)
=>Here data will be exchanged in binary format b/w consumer and producer

EJB

=>gives by Sun Ms
=> Enhancement of RMI
=>It is language dependent
=> It is platform and Architecture independent
=>can use both LAN (Local Arean network) and Internet(WAN) as communication channel
=> needs the heavy weight EJB Container to execute EJB Comps [EJB container is part of another heavy weight Applicaiton SErver like weblogic, glassFish and etc..)
=>Gives lots of built-in middlewares (It was popular for banking apps earlier for this reason)
=> Outdated becoz of webServices
=> It is technology of JEE module and EJB Containers of applicaiton server softwares will come as the implementations..
=>We must deploy the devleoped EJB comps in the EJB containers of Application server for execution..

note:: Though Tomcat is called as Applicaiton server from version 7 .. It is still not providing EJB container.

=> Very complex to lean and execute
=>Here data will be exchanged in binary format b/w consumer and producer

CORBA (Common Object Reqquest Broker Archicteture)

=> Platform , Architecture and language indepndent
=> we can develop/produce and consume services in multiple languages like java , c++ , c and etc..
=> CORBA is specification and it will be implemented as IDL (Interface definitation language)
=> CORBA is complex learn and apply
=> CORBA is heavy weight to implement
=> CORBA looks great conceptwise .. But gives problems towards the implementations.

Dec 01.1 Intro to Distributed Apps

WebServices

=>WebSErvices is a mechanism of linking two apps as consumer and producer apps using the protocol http .

=> WebSerevices in platform independent and archicture independent and language independent..

A producer developed in "Java" can be used/consumed in .net , java, php, phyton , Java script and etc.. (Even reverse possible)

=> WebSErvices says Develop/produce any where and consume any where.

=> Webservices makes the consumer and producer Apps exchanging data either in XML or in JSON (Global formats)

=> Java and languages have provided multiple apis/Technologies and frameworks to implement web service logics as skelton logics (Integration Logics Producer) and as stub logics (Integration logics of consumer)

=>WebServices support http request and http response based method invocation i.e as http request the cosumer App invokes method of producer App and the results method execution goes back to consumer App as http response.

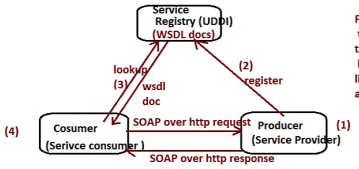
Two ways of implementing web services
- a) SOAP WebServices
- b) RESTfull WebServices

SOAP :: Simple object Access protocol (protocol)
REST :: REpresentational STatefull
web services (It is not a protocol it is kind of architecture/ mechanism)

a) SOAP WebServices

=> This runs based 3 component pricipile a) SErivce Provider(producer /Server)
b) Sevice Consumer (Consumer/Client)
c) Service Registry (where services will be registered to expose)

The Registries in SOAP based webServices env.. is UDDI (Universal Description Discovery and Integration)



For every SOAP web service that is registered in service Registry there will be one WSDL doc (xml document) having multiple details about web service like how to consume, the services names and etc (endpoint details)

WSDL :: WebService Description Language

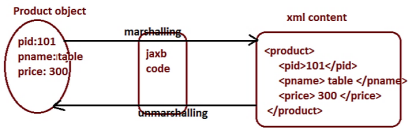
=> The http request carries xml baased SOAP message as request body nothing but inputs

=> The http response carries xml baased SOAP message as response body nothing but outputs

SOAP message strictly typed or complex XML messages which will be conveted java obejcts and reverse using Jaxb api

jaxb :: Java archicture for xml binding (nothing but converting java object data to xml content and vice-versa)

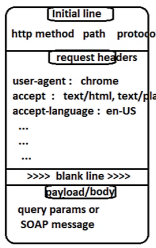
java object data to xml conversion is called marshalling and reverse is called unmarshalling



=>The consumer sends inputs to producer as SOAP message(xml) in http request
=>The Poducer sends outputs to cosumer as SOAP message(xml) in http response

=>The http request and http response contains two parts
- a) HEAD part
  - |---> contains two sections
    - a) Initial Line
    - b) headers
- b) Body part

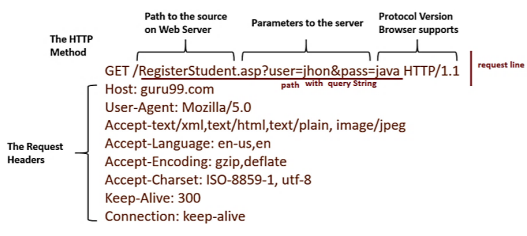
Structure of http request
=========



http methods or modes are (9)
- GET
- POST
- HEAD
- TRACE
- PUT
- DELETE
- OPTIONS
- CONNECT (reserved for future)
- PATCH (for partial Updates)

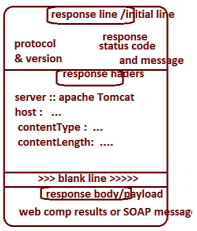
In Normal http requeust , the request body will be query String having request param names and values like sno=101&sname=raja&sadd=hyd
In soap over http request , the request body will be xml based SOAP message (xml tags)

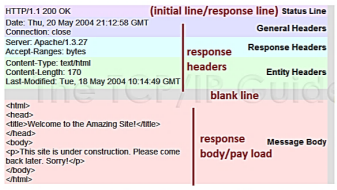
=>POST mode reuqest contain request body representing query String where as GET Mode request does not request body becoz its carries inputs as query String appended to the request url.



Http response status codes
==========
- 1xx (100-199) :: Informational
- 2xx (200-299) :: Success
- 3xx (300-399) :: Redirection
- 4xx (400-499) :: Incomplete (Client side errors)
- 5xx (500-599) :: Server side errors

Http response structure
==========

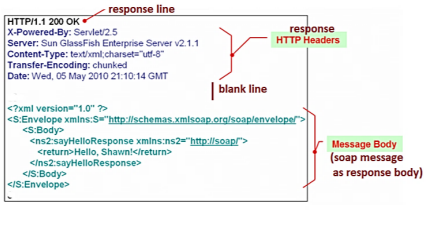




Normal http response contains the web comp genertad ouput (generally html code or plain text) as response body where as soap over http response contains the soap message(xml) as the response body

soap over http request
==========

SOAP over http response



Dec 02 Intro to Distributed Apps

WebServices
==========

=>WebSErvices is a mechanism of linking two apps as consumer and producer apps using the protocol http .

=> WebSerevices in platform independent and archicture independent and language independent..

A producer developed in "Java" can be used/consumed in .net , java, php, phyton , Java script and etc.. (Even reverse possible)

=> WebSErvices says Develop/produce any where and consume any where.

=> Webservices makes the consumer and producer Apps exchanging data either in XML or in JSON (Global formats)

=> Java and languages have provided multiple apis/Technologies and frameworks to implement web service logics as skelton logics (Integration Logics Producer) and as stub logics (Integration logics of consumer)

=>WebServices support http request and http response based method invocation i.e as http request the cosumer App invokes method of producer App and the results method execution goes back to consumer App as http response.

Two ways of implementing web services

a) SOAP WebServices — SOAP :: Simple object Access protocol (protocol)
b) RESTfull WebServices — REST :: REpresentational STatefull

web services (It is not a protocol it is kind of architecture/ mechanism)

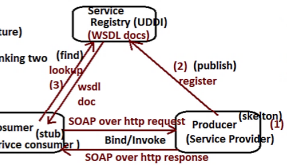
a) SOAP WebServices

=> This runs based 3 component priciple a) SErivce Provider(producer /Server)
b) Sevice Consumer (Consumer/Client)
c) Service Registry (where services will be registered to expose)

The Registries in SOAP based webServices env.. is UDDI (Universal Description Discovery and Integration)

SOAP based web services fall under SOA Archicteture (Service Orienteted Architecture)

=>SOA is the mechanism of linking two Apps contains
=> SOA service producer, Service consumer and Service registry as shown in the diagram



For every SOAP web service that is registered in service Registry there will be one WSDL doc (xml document) having multiple details about web service like how to consume, the services names and etc (endpoint details)

WSDL :: WebService Description Language

=> The http request carries xml baased SOAP message as request body nothing but inputs

=> The http response carries xml baased SOAP message as response body nothing but outputs

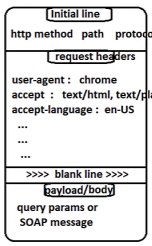
SOAP message strictly typed or complex XML messages which will be conveted java obejcts and reverse using Jaxb api

jaxb :: Java archicture for xml binding (nothing but converting java object data to xml content and vice-versa)

=>The consumer sends inputs to producer as SOAP message(xml) in http request
=>The Poducer sends outputs to cosumer as SOAP message(xml) in http response
=>The http request and http response contains two parts

a) HEAD part
|—> contains two sections
a) Initial Line
b) headers
b) Body part

Structure of http request
==========

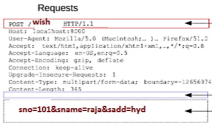


http methods or modes are (9)

GET
POST
HEAD
TRACE
PUT
DELETE
OPTIONS
CONNECT (reserved for future)
PATCH (for partial Updates)

In Normal http requeust , the request body will be query String having request param names and values like sno=101&sname=raja&sadd=hyd
In soap over http request , the request body will be xml based SOAP message (xml tags)

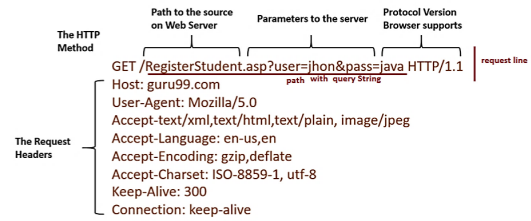
Requests



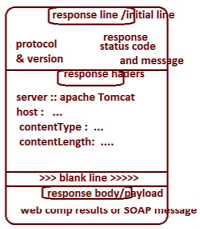
Http response status codes
==========

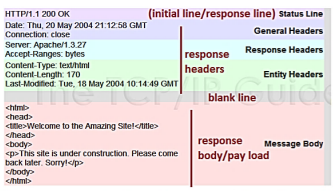
1xx (100-199) :: Informational
2xx (200-299) :: Success
3xx (300-399) :: Redirection
4xx (400-499) :: Incomplete (Client side errors)
5xx (500-599) :: Server side errors

=>POST mode reuqest contain request body representing query String where as GET Mode request does not request body becoz its carries inputs as query String appended to the request url.



Http response structure
==========

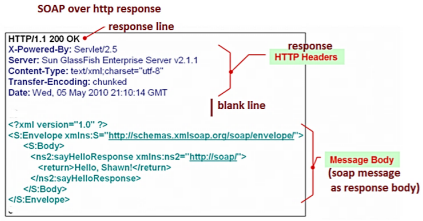




Normal http response contains the web comp genertad ouput (generally html code or plain text) as response body where as soap over http response contains the soap message(xml) as the response body

soap over http request
==========

SOAP over http response



In Java env... Jax-ws is api or technology (part of JSE module) that is given to developing SOAP based web services and the frameworks implementing SOAP based web services are a) apache axis b) apache cxf and etc..

Limitations of SOAP based web services
==========

a) These are based on SOA where seperate registry is required to publish/find web services ..
b) The service registries in SOA are very heavy maintain
c) The SOAP messages in http request and http response are complex to build and heavy to send over the network
d) Understanding about web service by reading WSDL document is not easy process.
e) SOAP message use complex Schema .. So building SOAP message is costly
f) Does not flexiblity to send data in different global formats .. should always use the heavy SOAP message as xml messages. and etc..

To overcome these problems use RestFull webServices.

Dec 02.2 Intro to Distributed Apps

b) RESTfull WebServices

=>REST is not a protocol .. REST is the mechasim or archicteture REpresenting the STate of web comp in http request and in http response So it is called RESTfull web services.

=> Allows to send data in both global formats XML , JSON

XML :: eXtensible Markup language
To convert Xml tags data to java object (unmarshalling) and reverse(marshalling) we use JaxB API (Java architecture for Xml binding)

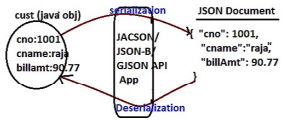
JSON :: Java Scripot Object Notation (nothing Object in Java script)

The process of converting Java object to JSON content (key: value pairs) is called Serialization and rerverse called DeSerialization

java data representation	java script data representation

String name="raja"
int age=30;
 	
Customer cust=new Customer();
cust.setCno(1001);
cust.setCname("raja");
cust.setBillAmt(90.77);
	var/let name="raja";
	var/let age=30;
	
	var/let cust= { "cno": 1001,
	"cname":"raja",
	"billAmt": 90.77
	}
	JSON doc means obj in Java script

To convert JSON content into Java Object and for reverse operation we use JSON APIs
eg: JACSON, JSON-B , GJSON and etc..



=> JSON is ligght weight compare to XML to represent the data ..

xml doc
```
<customers>
  <customer>
    <cno>101</cno>
    <cname>raja </cname>
    <billAmt>90.77 </billAmt>
  </customer>
</customers>
```

=>here compare to data,.. the structure is more like closing tag for every opening tag is bit extra

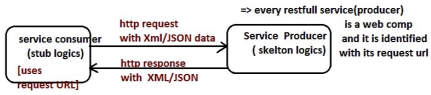
=>Gives structured data having relationship among the data memebers in bit heavy env..

JSON Doc
```
{ "cno": 1001,
  "cname":"raja",
  "billAmt": 90.77
}
```

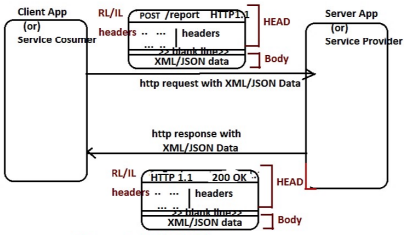
=>Here data and structure both are in good combination

=>Gives structured data having relationship among the data memebers in light env..

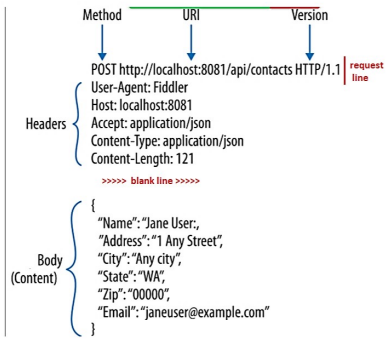
=> RESTfull is not based on SOA (SErvice oriented archicture) .. It is based on Consume- Producure Archicteture with FrontController support i.e no service registry is required here..



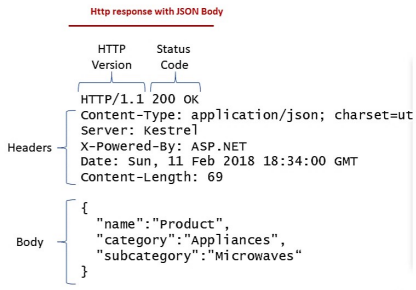
=> In RESTfull web services we prefer JSON data alot compare to xml data..
=> The service Producer (Restfull comp) can be developed in any language like java ,.net, php, phyton and etc..
=> The service consumer (Client) can be developed in any language like java ,.net, php, phyton and etc.. also can be as any app like Java app, andriod app, destktop app, php app, angular App, React App, IOS App and etc..



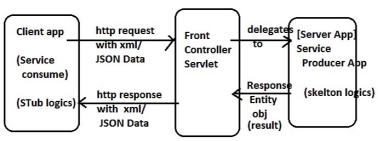
Http request with JSON BODY



Http response with JSON Body



=>The service provider in Restfull App is java class and it can not take http requests directly.. So we put one FrontController servlet like DispatcherServlet (spring) , ActionServlet(struts) and etc..
=> The special servlet that acts as entry and exit point for all requests and reponses is called Frontroller Servlet



=>In Java , Jax-RS is technology /api to develop Restfull web services where as Spring Rest , Rest easy , Restlet,Jersy and etc.. are the f/w to support Restfull web services implementation.

=>Spring Rest is given on top of spring MVC

Spring Rest =spring MVC ++

Summary on webservices

web services

SOAP Based web serivces (SOA) (Old/Legacy) ← Desiging Models → RestFull webSErvices (Consumer-Producer with FrontController) (Modren /Relatively New)

Jax-WS ← Programming APIs → Jax-RS

Apache Axis, Apache CXF ← Proramming Fraeworks → Spring Rest, Rest easy, Restlet, Jersy

SOAP over Http ← protocol → Http with JSON/Xml data

note:: Developing service providers in web services is also called as devleoping APIs.

note:: gathering and providing info that are required to develop service cosumer for invoking the services of spefic service provider is nothing but providing endpoints.

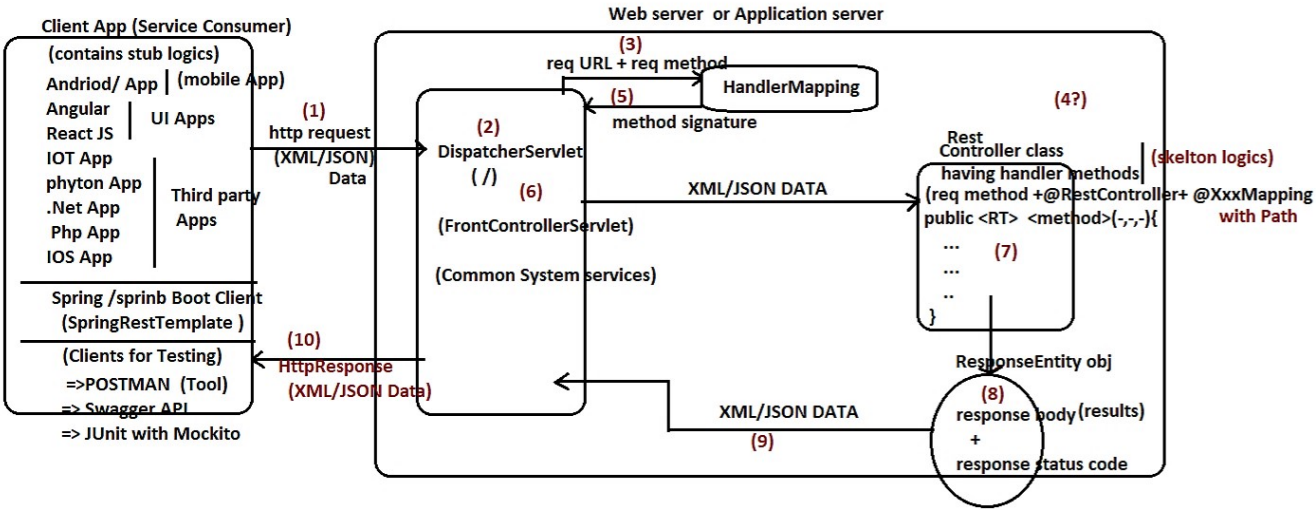
Dec 03 Spring Rest Apps

=>Spring Rest = spring mvc ++
That means spring mvc can be used to develop web application and also
RESTfull web service comps

=> Spring Rest is given based Consumer - Producer + FrontController Pattern
=> RESTfull webservices does not fall under SOA (Service Oriented Architecture) becoz there is not service registry in RESTFull Webservices .. More importantly the service provider comp of Restfull webSErvice is identified with it request url or path

=>FrontController is special web comp of web application that traps and takes either all or multiple http requests applies common system services and delegates requests to appropriate comps(controller classes) and gathers the results to send to clients.. Infact it act as entry and exit point for all the requests and responses of the web application.

Spring MVC Flow to towards developing RESTFul webservices (Spring Rest Flow)
==========



=> In RestFull web applications , there will not be any kind of view comps .. the methods of controller or RestController class directly send response to Client App through Dispatcher Servlet having XML/JSON Data which intially gernated as ResponseEntity object and later converted to XML/JSON Data

note:: In web application the client is browser window (Not programmable) giving request and getting response
note :: In Distributed Apps the Client/serivice cosumer is programmable App having logics to invoke methods of service producer(webSErvice comps) (Here browser can not be taken as client)

With respect diagram
==========

(1) The service cosumer App write stub logic by invoking service producer methods .. In this process Http request will be generated having XML/JSON Data

(2) The FrontController , DispatcherServlet (DS) traps and takes requests to apply common system services like logging, auditing and etc..

(3) The DS handovers the request to handler mapping comp giving request uri/path and request mode

(4?) The handler mapping comp searches for @Controller or @RestController classes for matching handler method and gets its signature using Reflection API

(5) HandlerMapping comp gives method signature to DS

(6) DS prepares necessary objects as required for the signature , gets controller or RestController obj and calls method

(7) Handler method executes either to process request directly or by taking the support of Service,DAO classes

(8) Handler method prepares HttpResponseEntity<T>object having response body(results) and http status code

(9) Handler method returns HttpResponseEntity<T> object to DS having XML/JSON data

(10) DS sends http resonse to Client App having XML/JSON data..

The methods of @RestController class does not need ViewResolver and View comps becoz they have natural behaviour of sending response to Client App directly through DipsatcherServlet.

Spring Rest = spring mvc ++ **Dec 06 Spring Rest 1st Apps**

=>Developing spring mvc handler method of controller class with out returning logical view name and making it returning ResponseEntity<T> object makes the handler method ready for Spring Rest env.. (Infact it becomes indirectly b.method of Restfull webservices)

=>To send response to browser /Client with out involving View .. we need to place @ResponseBody on the top of handler method in @Controller class.. Instread of writing two annotations we can just use @RestController.

@RestController = @Controller +@ResponseBody

eg1:

```
@Controller
@RequestMapping("/customer")
public class CustomerController{

 @ResponseBody
 @GetMaping("/display")
 public ResponseEntity<String> dispplayMessage(){
    ....
    ....
     ...
  return obj of ResponseEntity;
 }
}
```

(or)

eg2:

```
@RestController
@RequestMapping("/customer")
public class CustomerController{

 @GetMaping("/display")
 public ResponseEntity<String> dispplayMessage(){
    ....
    ....
     ...
  return obj of ResponseEntity;
 }
}
```

In spring MVC /Spring Rest the following cfgs takes place automatically

=> DispacherServlet cfg as FrontController with "/" url pattern
=> HandlerMapping (Default is RequestMappingHadlerMapping)
=> ErrorFilters dispplaying white label error pages
=> ViewResolver (default InternalResourceViewResolver)
and etc...

(we do not them mostly in Spring Rest programming becoz it does not use browser as client)

ResponseEntity<T> object contains two parts

a) Response body (String /object/collection)
b) Response Status code (we use ResponseStatus enum constants for like ResponseStatus.OK and etc.)

=>RepsonseStatus/Http Status indicates wheather the generated response/output/result should be given to Client as success output(200-299) or as Client Side error(400-499) or as server side error (500-599) and etc..

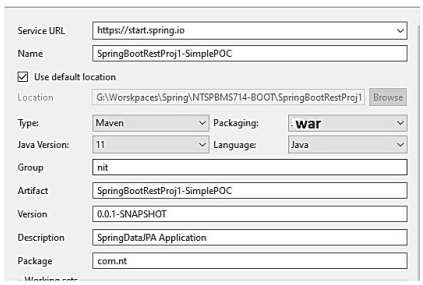
where as,

ResponseEntity<T> indicates (as the return type of hanlder method) that the generated ouput/result should go to client/browser directlty through DispatcherServlet with out involving any ViewResolvers and View Comps.

=>if ResponseEntity object is having other than <String> type as genertic type(object/collection) , then intenrally converts and holds output as JSON data (key-value) pairs.. Where as <String> type contains normal text content

Procedure to develope and Test First SpringRest App

step1) create spring boot starter project
adding spring web stater and taking packing as war file



-->next ---> select spring web starter ----> finish.

step2) Develop the @RestController class as shown below.

MessageRenderController *.java*

```
package com.nt.controller;

import java.time.LocalDateTime;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController  // (@Controller + ( @ResponseBody  on all the handler methods))
@RequestMapping("/message")  //Global path (purely optional)
public class MessageRenderController {

    @GetMapping("/generate")
    public  ResponseEntity<String>  generateMessage(){
        //get Sys date and time
        LocalDateTime ldt= LocalDateTime.now();
         //get current hour of the day  (response body)
        int hour=ldt.getHour();
        String body=null;
        if(hour<12)
             body="Good Morning";
        else if(hour<16)
            body="Good AfterNoon";
        else if(hour<20)
            body="Good Evening";
        else
            body="Good Night";
        // response status
        HttpStatus  status=HttpStatus.OK;
        //create ResponseEntity object
        ResponseEntity<String> entity=new ResponseEntity<String>(body, status);
        return entity;
    }//method
}//class
```

Dec 06.1 Spring Rest 1st Apps

step3) Test the app using browser or POSTMAN tool

note:: since ResponseEntity<String> is taken and handler method type is "GET" (in this controller)
we can send that request even from browser (otherwise not possible from browser)

Using browser (Not recomanded to use becoz it can send only GET or POST mode requests
========== where as @RestController can have GET ,POST, DELETE,PUT,PATCH and etc.. modes handler methods)

a) Run the Application using Run as server option
b) use the following from the browser

http://localhost:3030/SpringBootRestProj01-SimplePOC/message/generate

Using POSTMAN Tool (Recomanded to use) (A good tool to Test the Resftull web services /service providers developed in any language/technology /framework having capability generate different modes of request and ability recieve text/JSON/XML content based response)

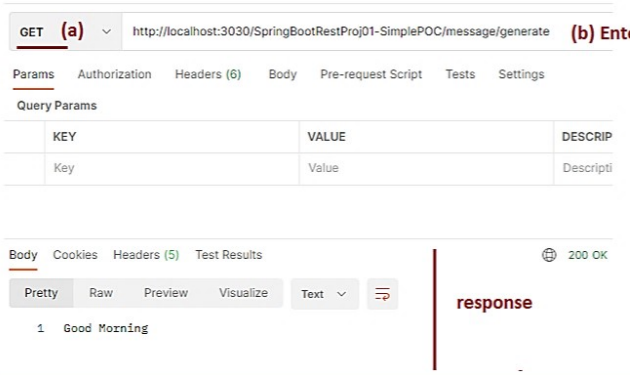
a) download and install postman tool
(by skipping the registration)

https://www.postman.com/downloads/ ----> use windows 32/64 bit for download

b) Create a new Collection
=> Postman home page ===> skip registration and go to the app (look at botton the page)===>
use (+) symbol to create the collection.

c) Send request by using the request url

Right click on the above created Collection --> Add Request --->



What is the difference between @Controller and @ResstController

@Controller	@RestController
a) Given in the spring 2.5 version (bit old annotation)	a) Given in the version spring 4.0 (relatively new annotation)
b) Specialization @Component	b) Specialization of @Controller
c) makes the java class webController class having capability to handle http requests by taking them through DispatcherServlet	c) makes the java class Rest Controller or Restfull service provider class having capability to handle http requests by taking them through DispatcherServlet
d) Can be used in both spring MVC and spring Rest Apps	d) Recomanded to use only in spring Rest Apps (Restful Apps)
e) Every Handler method does not get @ResponseBody automtically, So we need to add it expicitly	e) Every handle method automatically gets @ReponseBody So there is no need adding it seperately
of the f) Based on the return type handler method it decides wheather view comp should be invovled or not (if the return type is otherthan ResponseEntity<T> then it involves ViewResolver and view comp)	f) Makes the handler method sending its output/results as response to client direcly through DispagcherServlet out with involving ViewResolver and View comp

note:: Instead of comparing @RestController with @Controller .. plese use it as convient annotation given over @Controller providing easyness to developer Restfull Service providers (@Contrller + @ResponseBody)

Dec 07 Working with JSON Data

Http request methods/modes
==========
GET
POST
PUT
DELETE
OPTIONS
TRACE
PATCH
HEAD
CONNECT (Resrved for Future)

Generally we use the following Http request methods/modes in Restful applications while performing CURD Operations

GET —> for Read Operations (R) ->selecting records
POST —> for Create opertions (C) -> inserting records
DELETE —> for DELETE opertions (D) -> deleting records
PUT —> for Update opertions (U) -> modifying records
PATCH —> for Update opertions (U) -> modifying records

note: PUT for complete modification of the record.
PATCH for partial modification of the record.

```
public <RT> updateEmployee(Employee emp){
   ....                                          use  | PUT
   ...                                               | mode request
}
```

```
public <RT> updateEmployee Email(String newEmail){
   ....                                          use  | PATCH
   ...                                               | mode request
}
```

=>Since HEAD request mode HttpResponse does not contain body /output/results , So it can not be used for Read /Select Opertions
=> Since TRACE is given to trace/debug componets involved for the SUCCESS or FAILURE of request processing it can not be used for CURD Operations
=>Since OPTIONS mode request gives the possible Http request methods/modes that can be used to generate request to web comp .. its HttpResponse contains purely list Modes/methods that are possible to give request to web comp..

```
@WebServlet("/testurl")
public class TestServlet{

  public void doGet(req,res){
     ...
     ...
  }
}
```

if we give OPTIONS mode request to this web comps we get response as
Allow : GET,HEAD,OPTIONS

note : when we give OPTIONS mode in postman you can type urlmapping of GetMapping , PostMapping, DeleteMapping, PutMapping, PatchMapping then you don't get error and you'll get the success response of 2XX status. but you dont get the response body/result.

```
@WebServlet("/testurl")
public class TestServlet{

  public void doPost(req,res){
     ...
     ...
  }
}
```

if we give OPTIONS mode request to this web comps we get response as
Allow : POST,OPTIONS,

=>if the request url is not valid then we get 404 error (requested resource not found)
=> if the request url based web comp is not ready
to process given mode request then we get 405 error (method not allowed)
=> if the request fails in Authentication then we get 401 error
=> if the request fails in Authorization then we get 403 error (resource is forbidden)
=> if the web comp(servlet/jsp/producer) fails to instantiate for the given request then we get 500 error

=>if the return type of producure method is other than <String> generic in ResponseEntity object then the producer methods send JSON data along with the HttpResponse body

=>if the return type of producer method is <String> generic in ResponseEntity object then the producer methods sends text data along with the HttpResponse body

Every Restfull application /Project contains
a) server App /prodcuder App/ Service provider App (spring MVC App with @RestController with methods) | (also called Rest API) [The request url of @RestController and other required infomation for sending request are called End points]

b) Client App /Service Cosumer/ Consumer App

Angular
ReactJS
PHP
IOS
IOT Devices
.net
python
andriod
| (Programable client Apps)

POSTMAN
Swagger
| Tools for Testing

spring RestTemplate (spring basedClient) (Programmable Client Apps)

request headers vs req pameters

=>req headers are Client generated
inputs that go along with request
automatically having fixed names(header names)
eg:: accept , accept-language, user-agent,referer ,contentType and etc.

=>req params are enduser supplied values
as query String /form data ..req param names not fixed ..
and they are user-defined
sno=101&sname=raja&sadd=hyd
req params /query params

==> API development means Developing Spring Rest Server App/Service provider App

=> Giving API End points means providing request url and other related information to developers for developing client Apps/service consumer App for cosuming the services offered by Service provider

note:: The API/ services developed in SOAP based webservices can not consumed using Rest Client and vice-versa.

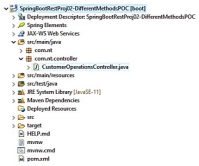
note: The Restful webSErvices developed using one kind of Rest API/framework (like jesry /spring rest, ja x-RS,Restlet and etc..) can be consumed using same Rest API or different Rest APIS (becoz both are in Restfull webService env..)

(public apis)
eg:: There are multiple open /free apis /service providers devleoped in different technologes/frameworks of Restfull webservices and they consumed in our Apps using our choice rest apis..

eg:: weather report api
Google Maps api
Gpay APIs
ICC API
Covid APIs
pay APIS

RestController class
==========

```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/customer")
public class CustomerOperationsController {

    @GetMapping("/report")
    public ResponseEntity<String> showCustomersReport(){
        return new ResponseEntity<String>("From GET-ShowReport Method" , HttpStatus.OK);

    }

    @PostMapping("/register")
    public ResponseEntity<String> registerCustomer(){
        return new ResponseEntity<String>("From POST-RegisterCustomer Method" , HttpStatus.OK);

    }

    @PutMapping("/modify")
    public ResponseEntity<String> updateCustomer(){
        return new ResponseEntity<String>("From PUT-UpdateCustomer() Method" , HttpStatus.OK);

    }

    @PatchMapping("/pmodify")
    public ResponseEntity<String> updateCustomerByNo(){
        return new ResponseEntity<String>("From PATH-UpdateCustomerByNo() Method" , HttpStatus.OK);

    }

    @DeleteMapping("/delete")
    public ResponseEntity<String> deleteCustomer(){
        return new ResponseEntity<String>("From DELETE-deleteCustomer Method" , HttpStatus.OK);

    }
}
```



GET http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/report Send

POST http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/register Send

DELETE http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/delete Send

PUT http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/modify Send

PATCH http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/pmodify Send

Dec 08 Working with JSON Data

Sending JSON Data as Http Response body to Client from service Provider App

=> if the method of Service Provider is returns ResponseEntity object having other than <String> generic (can be any other object) then given object will be converted to JSON data and will be placed in HttpResonse as body automatically..

JSON :: Java Script Object Notation

=>It is a way of representing Object data using key: value format

=> Key must be in "" and value can be any thing.. if the value is "string" content then it also should be in ""

=> one {} (flower bracket) represents one object or sub object

=> "[]" represents array/list/set element values

=> In JSON array/list/set collection will be treated as array only ,so their elements will be represented using [-,-,-]

=> In JSON array/list/set collection are called 1D arrays

=> In JSON map collection is called 2D array

=> In JSON Map collection elements and HAS-A property elements will be represented using sub object/node { "key":value , "key":value,....}

```
Customer cust=new Customer(1001,"raja","hyd",67877);

In Json::
    {
      "cno": 1001,
      "cname":"raja",
       "cadd" : "hyd",
       "billAmt": 67877
      }
```

cust (Customer obj)
cno:1001
cname:raja
cadd:hyd
billAmt:67877

=>HTML (Hyper Text Markup Language) is given to display data on browser by applying styles

=> JSON/XML are given to describe data (To construct data having structure)

What is the difference b/w JSON and XML?

YML , MongoDB docs are inspired from JSON

```
Student st=new Student(101,"jani",67.88);
```

st(Student obj)
sno:101
sname: jani
avg: 67.88

JSON way of representing data

```
{
 "sno":101,
 "sname":"jani",
"avg":67.88
}
```

(Both are Global format for representing Data)

Xml way of representing Data

```
..... //schema or DTD import (optional)

<student>
   <sno>101</sno>
   <sname>jani</sname>
    <avg> 67.88 </avg>
  </student>
```

JSON	XML
(a) It is Java Script Object Notation	(a) It is eXtensible Markup language
(b) It is the way of represnting object data	(b) it is way constructing /descring data using tags
(c) maintains th data as "key": "value" pairs { "key":"value", "key" : "vlaue" }	(c) maintains the data using tags as the tags body and attributes [<> </>]
(d) Does not Provides namespaces to validate the data	(d) provides namespaces to validate data (namespace is a library that contains set tags , attributes and rules to construct xml data)
(e) This format is given by Java Script laguage (Sun Ms + Netscape)	(e) This is given according to w3c recomadations and inspired from SGML (Standard Generalized Markup Language)
(f) JSON files are easy to read and process	(f) Xml files are bit complex to read and process.
(g) To handle JSON Data we use the support of JACSON api	(g) To handler Xml data we take support of xml apis like Jaxp (java api for xml processing) Jaxp deals with SAX api, DOM api, JDOM api and etc..
(h) To covert JSON data to Object and vice-versa we can use GSON , JSON-B and etc.. api	(i) To convert Xml doc to objects and vice-versa we use JaxB api (Java api for Xml Binding)
(i) The Process converting object to JSON Data is called serialization and reverse is called Deserilziation	(i) The Process of converting object to xml data is called marshalling and reverse is called unMarshalling
(j) JSON fomat/ files are ligh weight compare to XML format/ files	j) XML fomat/ files are bit heavy weight compare to JSON format/ files
(k) Very much used is Restfull web serices as alternate to XML to send and recive data	(k) Very much used in SOAP based web services to send and recieve data (SOAP message are xml messages)
(l) JSON is less structured compare to Xml	(l) Xml is more structured..
(m) does not support commenting	(m) supports commenting
(n) Less secured becoz it just contains key and values	(n) More secured becoz data is having hierachy strcuture and namespaces procted..
(o) Allows only UTF-8 Characters	(o) Allows lots of Charsets including UTF-8

Dec 08.1 Working with JSON Data

Making RestController methods (service methods) sending Java Object data as JSON data

```
//RestController
package com.nt.controller;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.nt.model.Customer;
@RestController
public class CustomerOperationsController {
    @GetMapping("/report")
    public ResponseEntity<Customer> showData(){
        Customer cust=new Customer(1001,"raja",54566.66f);
        HttpStatus status=HttpStatus.OK;
        return new ResponseEntity<Customer>(cust,status);
    }
}
```

```
//model class
package com.nt.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
    private Integer cno;
    private String cname;
    private Float billAmt;

}
```

from POSTMAN

GET http://localhost:3030/SpringBootRestProj03-SendingJSONDataAsResponse/report (SEND)

response body

```
{
   "cno": 1001,
   "cname": "raja",
   "billAmt": 54566.66
}
```

Sending Complext obj data as the complext JSON Data to Client from RestController

```
// In RestControler class
@GetMapping("/report1")
    public ResponseEntity<Customer> showData1(){
        //body
        Customer cust=new Customer(
                1001,"raja",54566.66f,
                    new String[] {"read","green","blue"},
                    List.of("10th","10+2","B.Tech"),
                    Set.of(544535345L,7576575L,6465654L),
                    Map.of("aadhar", 35345435L, "panNo", 354353534L),
                    new Company("SAMSUNG","hyd","Eletronics",4000));

        //status
    HttpStatus status=HttpStatus.OK;
    return new ResponseEntity<Customer>(cust,status);
  }
```

```
package com.nt.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Company {
  private String name;
  private String addrs;
  private String type;
  private Integer size;
}
```

Model class

Customer.java

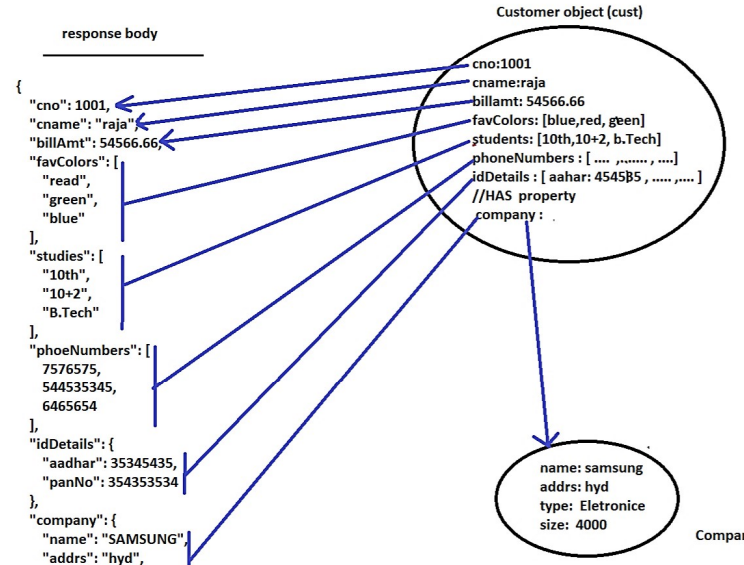
```
package com.nt.model;

import java.util.List;
import java.util.Map;
import java.util.Set;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
    private Integer cno;
    private String cname;
    private Float billAmt;
    private String[] favColors;
    private List<String> studies;
    private Set<Long> phoeNumbers;
    private Map<String,Object> idDetails;
    //HAS- A property
    private Company company;

}
```

POSTMAN Tool

GET http://localhost:3030/SpringBootRestProj03-SendingJSONDataAsResponse/report1 Send

response body

```
{
  "cno": 1001,
  "cname": "raja",
  "billAmt": 54566.66,
  "favColors": [
    "read",
    "green",
    "blue"
  ],
  "studies": [
    "10th",
    "10+2",
    "B.Tech"
  ],
  "phoeNumbers": [
    7576575,
    544535345,
    6465654
  ],
  "idDetails": {
    "aadhar": 35345435,
    "panNo": 354353534
  },
  "company": {
    "name": "SAMSUNG",
    "addrs": "hyd",
    "type": "Eletronics",
    "size": 4000
  }
}
```

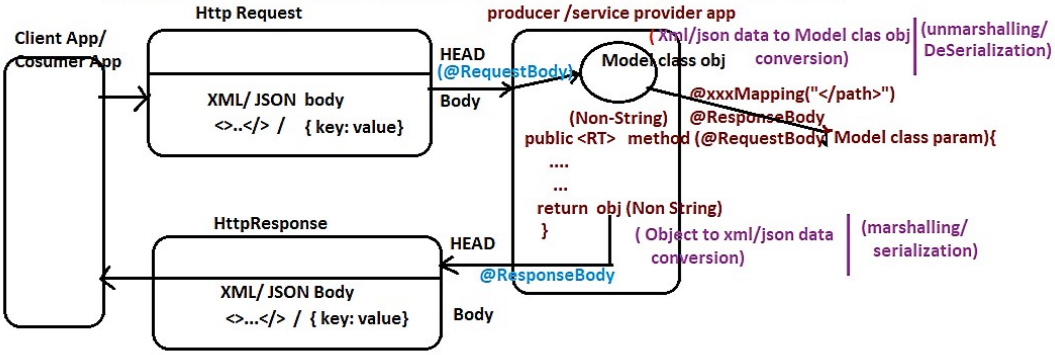


Dec 10 Response as XML or JSON Data

Working with Media Type Annotations (@RequestBody and @ResponseBody)

=>Using @RequestBody for the service provider/producer method param we can convert given xml or JSON data into given Model class (java bean class obj).

=>Using @ResponseBody on the top of service provider/producer method we can convert non-String content into xml or JSON data and can send to client as response body.
(@ResponseBody will be applied automatically on every method of @RestController class)



Making Spring Rest Producer App Sending XML Response body to Client App (Consumer App)

=> Based on given request type, making producer App sending different type of content with content type is techically called Content negotiation.

=> For this we need to use "Accept" request header specifying content type like "JSON", "XML" and etc.. This make producer application given content in response in the requested format ..
Though "Accept" is request header , it makes the Producer/server/service provider App sending its content in required format like JSON, XML and etc..

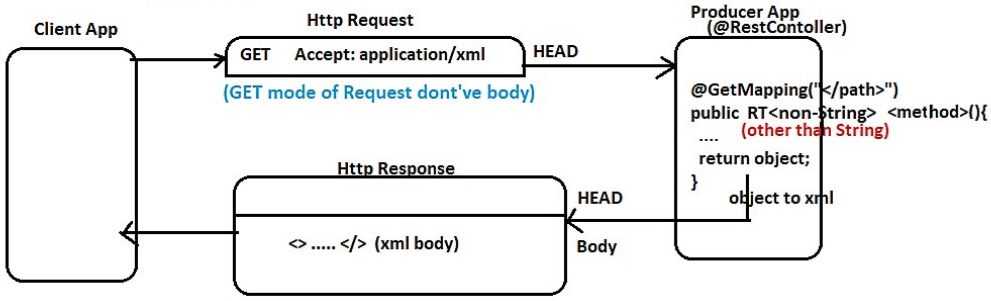
=>By default every method in Producer App sends palin text response if the return type is ResponseEntity<String> otherwise (for RespinseEntity<non-String> it sends JSON response)

Making Producer Application Sending XML response

=>For this the Client app/consume App must set "Acccept" request header to "applicaiton/xml" (default is */*)
Accept: application/xml tells to producer App give me response content as xml content
Accept: */* tells to Producer Apps give the response in any format (what u want, that u can give)

=> We need to add the following addtional depenency(jar file) to the project for converting Object data to xml format

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-
dataformat-xml -->
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.12.5</version>
</dependency>
```



Example App

step1) create Spring Boot starter project having following starters, dependencices (spring web , lombok, jackson-dataformat-xml)
(starters) (dependencies)

step2) Develioe the Model class

```
//Model class
=============
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
      private Integer cno;
      private String cname;
      private String cadd;
      private Float billamt;
}
```

Dec 10.1 Response as XML or JSON Data

step3) Develop Producer as @RestController having method with other String Return type.

//Producer App

```
package com.nt.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.nt.model.Customer;
@RestController
public class CustomerOperationsController {

    @GetMapping("/report")
    public Customer showData() {
        Customer cust=new Customer(1001,"raja","hyd",44545.77f);
        return cust;

    }
}
```

not a recommended approach and moreover Default response status code 200 will be taken

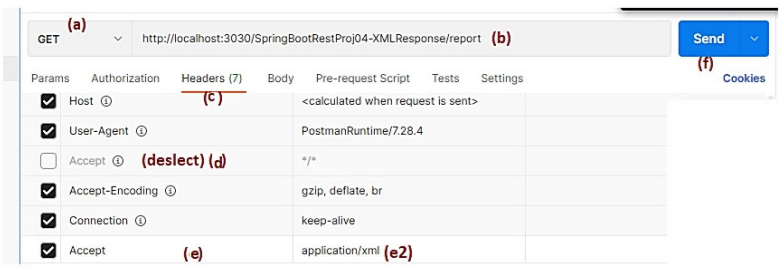
(or)

```
@GetMapping("/report")
    public ResponseEntity<Customer> showData() {
        Customer cust=new Customer(1001,"raja","hyd",44545.77f);
        return new ResponseEntity<Customer>(cust,HttpStatus.OK);

    }
```

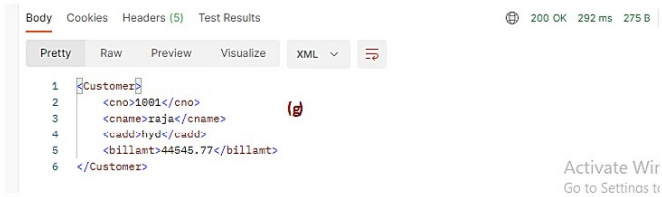
Taking ResponseEntity<T> as Return is recomanded becoz it gives control on response status code.

step4) Run the Applicaiton
(RunAs --- Run on server)

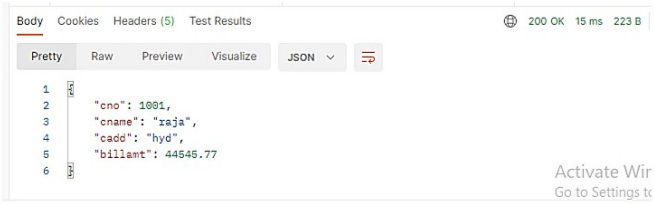
step5) Give Request from POSTMAN Tool by setting "Accept" header with the value "application/xml"



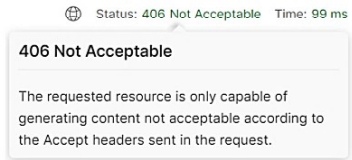
Response



=>if we send the above request with out placing Accept: application/xml request header then Accept:*/* will be take as default value and we get Json response as show below

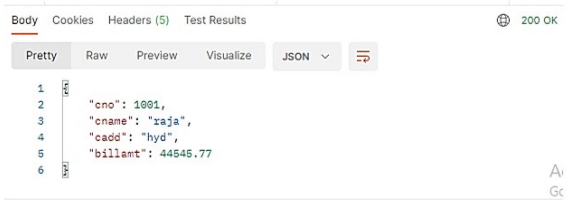


=>In the above setup where we asking for XML response using "Accept:application/xml" request header , if we are not placing "jackson-dataformat-xml" then we get 406 (Media Type Not Acceptable) Error



=> we run the same Application with the above setup (not adding jackson-dataformat-xml jar file) and with out chaging "Accept" header value to "applicaiton/xml" then we get JSON response fron producer App becoz the method return non-String type

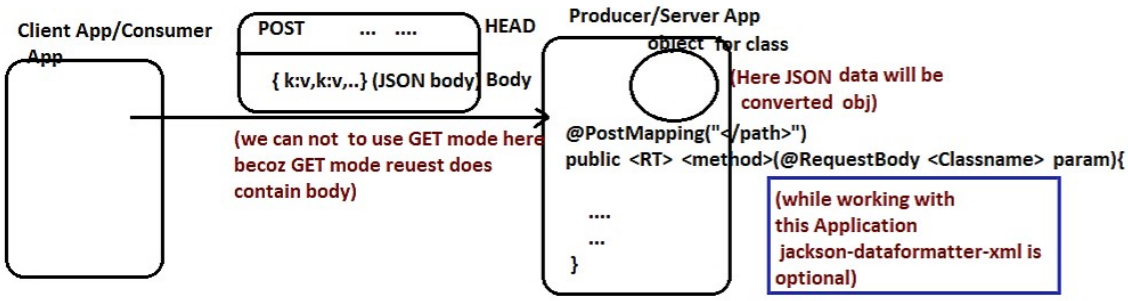
note:- Becoz giving the response in the JSON format is the default



Dec 11 Working with JSON - XML Data with @RequestBody

Working with @RequestBody to convert given XML/JSON content into Model class obj

=> if Client generated http request is having XML/JSON content as the body and if want to construct java class object having that data then we need to specify @RequestBody for the parameter of Producer method whose type is Model class name/Java class name



Syntax to use @RequestBody in method param

```
@RequestBody <ClassName> <paramName>
```

becomes obj name internally

=>GET, HEAD mode request does not contains body part.. but they send little amount of data as request param names and values in query String

=> The response of GET mode request contains both HEAD , BODY part where the response of HEAD mode request does not contains BODY part .. So we use GET mode request to get data from server and HEAD mode request to check wheather certains web comp is avaiable or not (Debugging purpose)

=> if JSON body is comming as { } (Empty folower bracket) then the Object for class given in @RequestBody will be ceated using 0-param constructor with default values given JVM (0/0.0/ null/false)

=> The keys in JSON body and the Property names in the give class of @RequestBody must match in order to see Data binding to object.. if few keys and property names are matching and others not matching then binding will happen only for few properties the remaining properties hold default values.

Example App

step1) Create spring Boot Project
adding the following starters , dependendencies
a) Lombok b) spring web c) jackson-dataformat-xml

note:- here for the dependency of jackson-dataformat-xml version not required

step2) Develop the Model class

```
//Model class
package com.nt.model;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
      private Integer cno;
      private  String cname;
      private  String cadd;
      private  Float billAmt;
}
```

step3) Devleop Producer App as @RestController

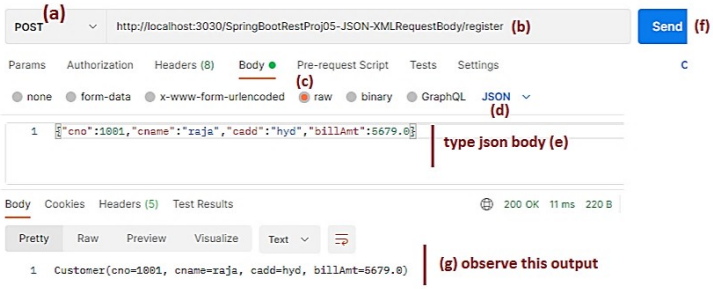
```
package com.nt.controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.nt.model.Customer;
@RestController
public class CustomerOperationsController {
      @PostMapping("/register")
      public  String  registerCustomer(@RequestBody  Customer cust) {
            return cust.toString();
      }
}
```

@RequestBody of spring Rest is very much similar to @ModelAttribute(-) of spring MVC

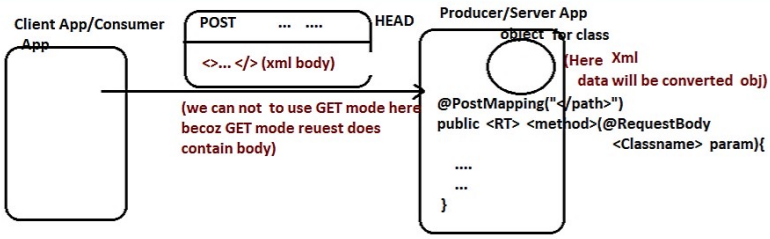
step4) Run the Application

Dec 11.1 Working with JSON - XML Data with @RequestBody

step5) Send the following request from POSTMAN



=> After taking request body as JSON/XML and if we do not send correct format body content along with the HttpRequest then we get 400 error response (Bad Request)

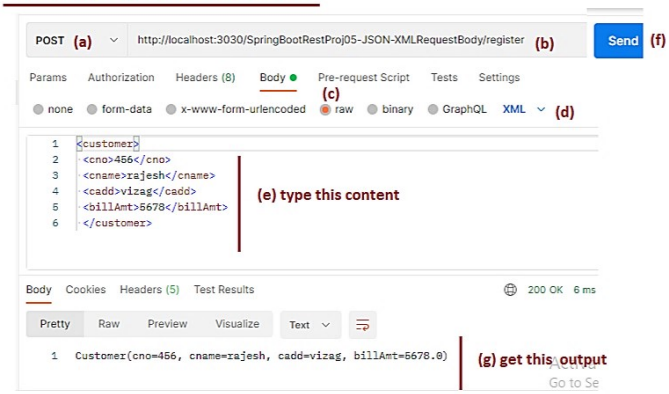


(while working with this Application jackson-dataformatter-xml is mandatory)

Example App (same as above)

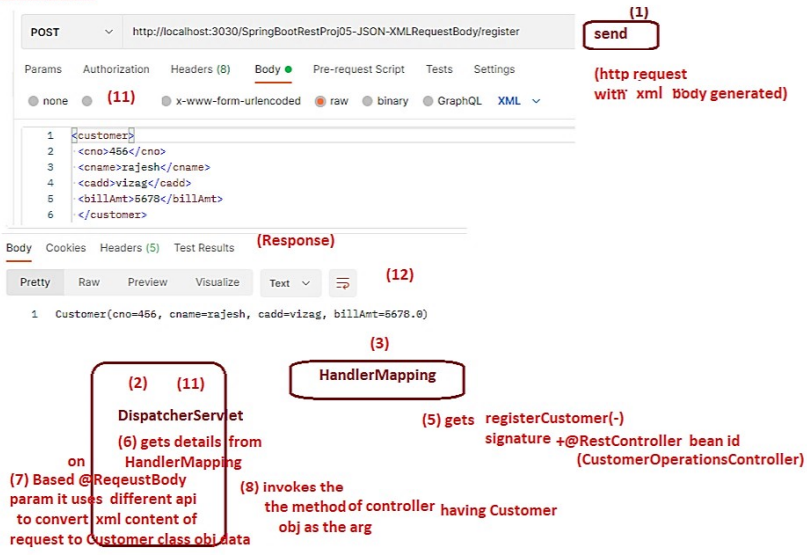
==========

Test the App using Post Main as show below



Flow of execution

==========




```
public class CustomerOperationsController {
                                                    (4?) searchê for Post mapping method with /register as reuest path
    @PostMapping("/register")   (9)
    public String registerCustomer(@RequestBody Customer cust) {
        return cust.toString();  (10)
    }
```

=> if request body content type is XML and the producer is not having jackson-datafromatter-xml iar file in the build path then we get 415 error (415 Unsupported Media Type)

conclusion

==========

406 :: we want Xml response through "Accept:application/xml" but to get jackson-dataformatter-xml jar is not added in the producer application (related @ResponseBody)

415 :: we want to send xml body along with request to store into Object using @RequestBody but jackson-dataformatter-xml jar is not added in the producer application

Dec 14 Working with JSON - XML Data with @RequestBody

Converting JSON body of Http request to Java class obj using @RequestBody

=>@ReqeustBody and @ResponseBody Annotations are called Media type annotations becoz they are useful to decide media of the incoming and outgoing content

=>@RequestBody is useful to convert JSON/XML data to Java class object and @ResponseBody is useful to convert Non-String data to JSON/XML data

=>@RequestBody dealing 1D and 2D Arrays JSON

1D array:: array/List/Set Collections "<variable>": [<val1>,<val2>,<val3>,...]

2D array :: Map "<variable>":{"<key1>":<val1>,"<key2>":val2,...}

2D Array :: HAS-A property "<variable>":{"<subProp1>":<val1>,"<subProp2>":val2,..}

Customer.java

```
package com.nt.model;

import java.util.List;
import java.util.Map;
import java.util.Set;

import lombok.Data;

@Data
public class Customer {
	private Integer cno;
	private String cname;
	private String[] favColors;
	private List<String> academics;
	private Set<Long> phoneNumbers;
	private Map<String,Double> billDetails;
	private Address addrs;

}
```

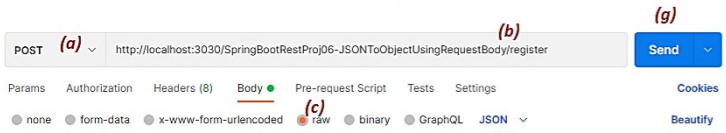
Address.java

```
package com.nt.model;
import lombok.Data;

@Data
public class Address {
	private String houseNo;
	private String streetName;
	private String location;
	private Long pinCode;

}
```

//Rest Controller class

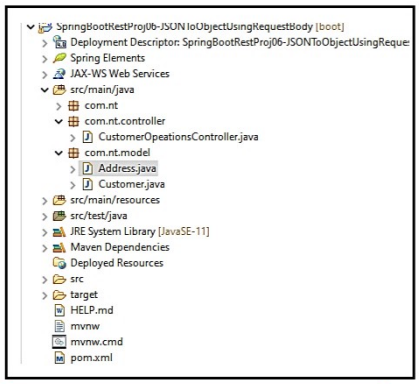
```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.nt.model.Customer;

@RestController
public class CustomerOpeationsController {

   @PostMapping("/register")
   public ResponseEntity<String>  saveCustomer(@RequestBody Customer cust) {
	return new ResponseEntity<String>(cust.toString(), HttpStatus.OK);
   }
}
```

run as run on server




```
{
   "cno": 101,
   "cname": "rajesh",
   "favColors": [
      "red",
      "blue",
      "gree"
   ],
   "academics": [
      "10+2",
      "B.Tech",
      "M.Tech"
   ],
   "phoneNumbers": [
      5535353435,
      54353455,
      5353533543
   ],
   "billDetails": {
      "x-mas tree": 6546.77,
      "cake": 5345.5,
      "chocolates": 6789.6
   },
   "addrs": {
      "houseNo": "1-2-4/566",
      "streetName": "RKStreet",
      "location": "hyd",
      "pinCode": 522345
   }
}
```

(e) type this cotent



Note:- When you use List of Map objects then in json type like this

eg: List<Map<String,Integer>> ages;

```
"ages":[{
        "age1":"28",
        "age2":"29"
      }],
```

wait for this outoput

Dec 14.1 Working with JSON - XML Data with @RequestBody

Converting JSON Data to Java class object using @RequestBody dealing with List<Object>, date, time values

For List<Object> or Collection<Object> we need to take

```
"<variable>": [ {"key":value,"key":value,...} ,
               {"key":value,"key":value,...},
               {"key":value,"key":value,...}
               ]
```

for List of Map objects:-

in java class:-- List<Map<String,Integer>> xyz;

in Json of Postman:-

```
"<var name>":[{ "key1":"val1",    "key2":"val2"  }],
```

Company.java

```
package com.nt.model;

import lombok.Data;

@Data
public class Company {
    private String name;
    private String location;
    private Integer size;

}
```

application.properties

```
server.port=4040
server.servlet.context-path=/JSONToJavaObjectApp1
```

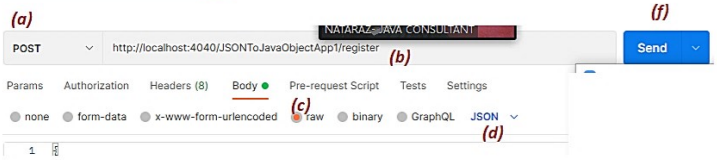
Customer.java

```
package com.nt.model;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Map;

import com.fasterxml.jackson.annotation.JsonFormat;

import lombok.Data;

@Data
public class Customer {
    private Integer cno;
    private String cname;
    private List<Company> companiesList;
    @JsonFormat(pattern="yyyy-MM-dd")
    private LocalDate dob;
    //@JsonFormat(pattern="yyyy-MM-dd HH:mm:ss")
    private LocalDateTime purchaseDate;
    private List<Map<String,Long>> familyIds;

}
```

//RestController

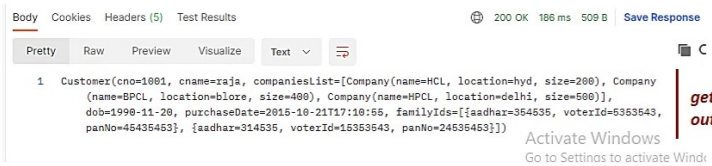
```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.nt.model.Customer;

@RestController
public class CustomerOpeationsController {

    @PostMapping("/register")
    public ResponseEntity<String>  saveCustomer(@RequestBody Customer cust) {
        return new ResponseEntity<String>(cust.toString(), HttpStatus.OK);
    }
}
```

Note:- Run this application as spring Boo App




```
{
 "cno":1001,
 "cname":"raja",
 "companiesList":[
   {"name":"HCL","location":"hyd","size":200 },
   {"name":"BPCL","location":"blore","size":400 },
   {"name":"HPCL","location":"delhi","size":500 }
 ],
 "dob":"1990-11-20",
 "purchaseDate":"2015-10-21 17:10:55",
 "familyIds":[
   {"aadhar":354535,"voterId":5353543,"panNo":45435453},
   {"aadhar":314535,"voterId":15353543,"panNo":24535453}
 ]
}
```

(e) type this content



get this output

---

note:: if the request body contains invalid JSON Pattern content which can not be parsed by JSON parser then the RestController sends 400 (Bad Request) error response to browser.

Assingment ::

=> Build Json format body for the class Structure

```
class E_Commerce{
  private String name;
  private String addrs;
  private Integer size;
  private List<Courier> couriers;
  private Set<PaymentGateway> gatewaysInfo;
   ...
   ..
}
```

```
class Courier{
  ...
  ...
}
```

```
class PaymentGateway{
  ...
  ...
  ...
}
```

Dec 15 Passing RequestParam to SpringRestApp

Passing request Params to Spring Rest App

=>The GET mode request does not contain body .. So the data we want to send in GET mode request should always be in the form of Query String

=>The GET mode request can carry limited amount of data (max of 2KB) that to as Query String params

=> In other mode requests (like POST,PUT,DELETE and etc..) the content in the request goes to server in the form of request body.

For Spring RestController we can pass data in GET Mode request in two ways

a) As request Params/ Query param in the Query String of request URL
(eg: url?key1=val1&key2=val2&key3=val3
(Supported by Spring MVC and Spring Rest)

b) As Path variable values in the requst URL
(Supported by Spring Rest .. not in Spring MVC)
(eg: url/value/value/value

{key} will be given in request path of b.methods placed @RestController.

note:: All web technologies /frameworks (Both java and Non-Java) supports request params as the BASIC concept web programming
(eg: servlet,jsp,php,asp.net , nodejs, express js and etc..)

note:: Passing values in the request URL as PATH variable values in introduced in RestFull progrmaming , So the all tehcnologies and frameworks supporting the Restfull programming (both java and non-java) gives provision to work with this path variable concept

a) Request Params/ Query Params in query String

=>To place then in request url ,we need
URL?key1=val1

=>To read in the method of @RestController
@RequestParam("key") paramType paramName

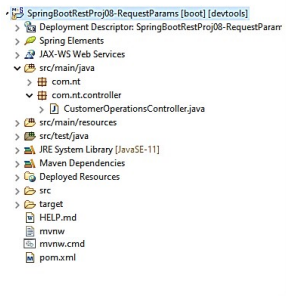
in the request url we must use "key" instead of "paramName"

(or) if you use "paramName" by having sprcific "key" you'll get 400 bad request

@RequestParam ParamType paramname

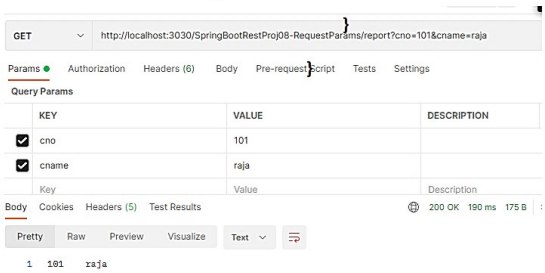
in the request url we must match with key of the request parameter

Example App



RestController class

```
package com.nt.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CustomerOperationsController {

 @GetMapping("/report")
 public String reportData(@RequestParam("cno") Integer no,
                          @RequestParam String cname) {
      return no+"   "+cname;
 }
}
```



=>while using request params in query String we can change the order of passing req param values

report?cname=raja&cno=101
(or)
report?cno=101&cname=raja

Both are correct

=> while using request params in query String if we pass addtional params that expected that will not generate any eror

?cname=raja&cno=101&cadd=hyd
Extra but does not generate error

=>when we pass queryString to the url in POST mode request.. they internally become request body content becoz the POST carries data as request body.

```
@RestController
public class CustomerOperationsController {
     @GetMapping("/report")
     public String reportData(@RequestParam("cno") Integer no,
                              @RequestParam(required = false) String cname) {
          return no+"   "+cname;
     }
}
```

URL :: http://localhost:3030/SpringBootRestProj08-RequestParams/report?cno=101
gives 101 NULL as response

```
@RestController
public class CustomerOperationsController {

     @GetMapping("/report")
     public String reportData(@RequestParam("cno") Integer no,
                              @RequestParam(required = true) String cname) {
          return no+"   "+cname;
     }
}
```

(default value of required param is true)

URL :: http://localhost:3030/SpringBootRestProj08-RequestParams/report?cno=101

=>Gives 400 bad request

Dec 15.1 Passing RequestParam to SpringRestApp

Send data as Path variable values

==========

=>Supports to minimise characters in request url while sending data

=> No need of passing seperate query String in the request URL to send data, we can pass data directly in the request url it self

=>passing "&,=" as the values in request param values is not possible .. but can be done easily using path variable

=> request URL with query String is not Clean url becoz it needs more chars (becoz both keys and values are required) and queyString needs seperate syntax tp fallow

=> Rquest URL with path variable values are part of request url itself and no need of following seperate syntax for it.

syntax :: rquest url (or) Path/<value1>/<value2>/<value3>/ .....

To read path variable values in @RestController methods

{key} for those values will give at @RestController in XxxMapping(...) method while defining the request path

@PathVariable Datatype paramname (or) @PathVariable("key") dataType param name → param name must match {key} of request path

here we use this "key" only in the path of @GetMapping of Java class instead of "paramName" without following this rule if you proceeds to next level at the time of sending data will get 500 error

=>The request path contains two parts while working path variables

a) static path (fixed path) (/<path>)

b) Dynamic path (key name whose value comes from request url) (/{key})

eg:: GetMapping("/report/{no}/{name}")

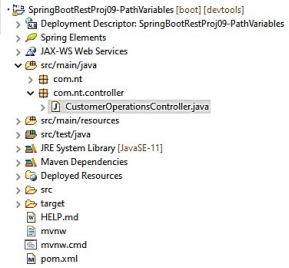
static path: /report — Dynamic path: /{no}/{name}

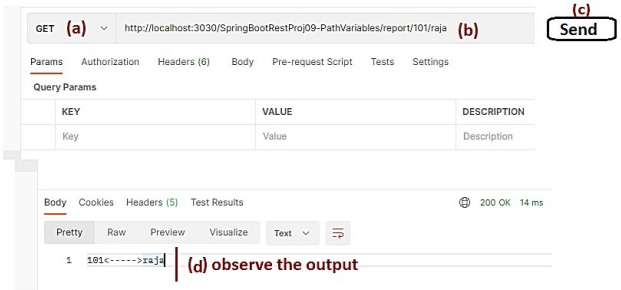
example rquest url

http://localhost:3035/SpringRestProj9-PathVariables/report/101/raja

static path: /report — path variable values (Dynamic path values): /101/raja

```
//RestController
================
package com.nt.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CustomerOperationsController {

    @GetMapping("/report/{no}/{name}")
    public String fetchData(@PathVariable("name")String cname,
                            @PathVariable Integer no) {
        return no+ "<----->"+cname;
    }

}
```





While working with request param we can change the order of passing their values

/report?cno=101&cname=raja

is same as

/report?cname=raja&cno=101

While working parth variable values we can not change the order of passing values

request path in @RestController method is :: GetMapping("/report/{no}/{name}")

/report/101/raja --> valid

/report/raja/101 ---> invalid -->Gives 400 Bad request if @PathVariable Integer no is taken

```
@GetMapping("/report/{no}/{name}")
    public String fetchData(@PathVariable("name")String cname,
                            @PathVariable Integer no) {
        return no+ "<----->"+cname;
    }
```

Raises Error

/report/raja/101 ---> valid --> if both params are taken as String values but the wrong will be stored in method params so the b.logic will be distrubed

```
@GetMapping("/report/{no}/{name}")
    public String fetchData(@PathVariable("name")String cname,
                            @PathVariable String no) {
        return no+ "<----->"+cname;
    }
```

Does not rais error but cname holds 101 and no holds raja which may distrub b.logics execution.

=>Giving extra path variable values (nothing but extra words path) in request url results 404 error

request path for @RestController method :: /report/{no}/{name}

request url :: http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/raja/hyd/india

Extra dynamic values which are not required So 404 error will come

Dec 16 Path Variables

Working with Path Variables
==========

=>if we give more or less values as path variable values than expected then we get 404 error (requested resource is not found)

eg::

```
@RestController
public class CustomerOperationsController {

    @GetMapping("/report/{no}/{name}")
    public  String  fetchData(@PathVariable("name")String  cname,
                              @PathVariable Integer  no) {
        return  no+ "<----->"+cname;
    }
}
```

request url :: http://localhost:3030/SpringRestProj10/report/101/raja

gives 101<-----> raja

request url :: http://localhost:3030/SpringRestProj10/report/101/raja/hyd

404 error

request url :: http://localhost:3030/SpringRestProj10/report/101

404 error

request url :: http://localhost:3030/SpringRestProj10/report

404 error

```
@RestController
public class CustomerOperationsController {

    @GetMapping("/report/{no}/{name}")
    public  String  fetchData(@PathVariable(name="name",required = false)String  cname,
                              @PathVariable(required = false) Integer  no) {
        return  no+ "<----->"+cname;
    }

}
```

request url :: http://localhost:3030/SpringRestProj10/report/101

we expect 101 <---> null should come as the output but
we get 404 error becoz no.of levels in request path are 3 and we are giving only two ,So it says requested resource is not found

request url :: http://localhost:3030/SpringRestProj10/report,

we expect null <------> null but we get
404 error (requested resource is not found)

---

if multuple methods are having similar request paths having same no.of levels then the request that is having more static level matchings will get priority

```
@RestController
public class CustomerOperationsController {

    @GetMapping("/report/no/name")
    public  String  fetchData1(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData1";
    }

    @GetMapping("/report/no/{name}")
    public  String  fetchData2(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData2";
    }

    @GetMapping("/report/{no}/{name}")
    public  String  fetchData3(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData3";
    }

    @GetMapping("/report/{no}/name")
    public  String  fetchData4(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData4";
    }
}
```

http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/name
output :: Fetch Data4

http://localhost:3030/SpringBootRestProj09-PathVariables/report/no/name
output :: Fetch Data1

http://localhost:3030/SpringBootRestProj09-PathVariables/report/no/rajesh
output :: Fetch Data2

http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/rajesh
output :: Fetch Data3

Dec 16.1 Path Variables

```
@RestController
public class CustomerOperationsController {
	@GetMapping("/report/101/raja")
	public  String  fetchData1() {
		return  "from FetchData1";
	}

	@GetMapping("/report/{no}/{name}")
	public  String  fetchData2(@PathVariable(name="name",)String  cname,
					@PathVariable Integer  no) {
		return  "from FetchData2";
	}
}
```

url :: http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/raja

output: fetchData1

url :: http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/ramesh

output: fetchData2

---

if two methods of @RestController is having same request path with same no.of levels then there is possibility of getting IllegalStateException during the application startup

```
@RestController
public class CustomerOperationsController {

	@GetMapping("/report/101/raja")
	public  String  fetchData1() {
		return  "from FetchData1";
	}

	@GetMapping("/report/{no}/{name}")
	public  String  fetchData2(@PathVariable(name="name",required = false)String  cname,
					@PathVariable(required = false) Integer  no) {
		return  "from FetchData2";
	}

	@GetMapping("/report/101/raja")
	public  String  fetchData3(@PathVariable(name="name",required = false)String  cname,
					@PathVariable(required = false) Integer  no) {
		return  "from FetchData3";
	}
}
```

gives

Caused by: java.lang.IllegalStateException: Ambiguous mapping.
Cannot map'customerOperationsController' method

this method also gives on the specified url path

Caused by: java.lang.IllegalStateException: Ambiguous mapping.
Cannot map'customerOperationsController' method

What is the difference b/w Request params and path variables way of passing data in spring Rest App ?

Request params	Path variables
a) Syntax to pass data along with request url is url?key=val&key=val&.....	a) Syntax to pass data along with request url is url/<static path>/val1/val2/... {key}/{key}/.. will be defined in @RestController class
b) syntax to read request param values is @RequestParam datatype param (or) @RequestParam("key") datatype param	b) syntax to read path variabe values is @PathVariable datatype param (or) @PathVariable("key") datatype param
c) while passing request param values in the query string of request url the order need not to match we	c) while passing path variables values in the request url the order must be matched
d) if pass more than required request params in query String the we will not error	d) if we pass more than required path variable values in the request url then we get 404 error
e) if we pass less than required request params in query Strung then we get error only when required=false is not taken @RequestParam	e) if we pass less than required path variable values in the request url then we get 404 error
f) Does not allow to pass '&' as value direct value (we vmust use %26 for that)	f) Does not allows "/" as value
g) Supported by both spring MVC and Spring Rest	g) Supported only in Spring Rest
h) In all web technologies that are there to develop web applicaitons in different domains supports this feature basic concept to pass data	h) supported only in Restfull programming of all domains..
i) URL is not clean URL (more characters required in the url to pass data)	i) URL is clean URL (less characters required in the url to pass data)
j) Easy to read and interpet	j) complex to read to interpet.
k) Bit slow while sending data	k) Bit faster while sending the data
l) Recomaned to use in non RestFull Apps	l) very useful in Restful apps..
m) Generally used while working GET mode requests becoz data goes as query String	M) works with all modes of of request becoz data goes directly from url itself

Dec 17 MiniProject on Spring Rest

Mini Project using Spring Rest + Spring data JPA + Oracle + POSTMAN tool

=> The methods is @RestController are called b.methods or opertaions (best) or web operations
=> The opeations of @RestController can have flexible signatures.
=> we can place b.logic directly in the operations of @RestController .. but its recomanded to place in service class methods and invoke them from @RestController operations
=> Developing @RestController with multiple operations linked with Service ,DAO class methods performing CURD Operations is nothing but developing Server /Service Provider/ Rest API/ API/ Producer .

=> Getting API and its operation details nothing but Basic URL + request path + path variables info together is called gathering API and its end points

API development /Producer/Server /ServiceProvider

our Layered App

(UI to send req)
POSTMAN Tool --------> (Front Controller) DispatcherServlet ---> (Exception handling logic) RestController ---> (b.logic) Service class ---> (persistence logic) DAO class ----->DB s/w (repository)
(UI to get resp)

[Tool based consumer)

step1) Create spring Boot project adding
the following staters oracle driver, spring data jpa,
spring web , devtools, lombok api

step2) Develop Entity Class /Model class that is required in o-r mapping operations

Tourist.java

```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@RequiredArgsConstructor
@Entity
@Table(name="REST_TOURIST")
public class Tourist {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer tid;

    @Column(length = 20)
    @NonNull
    private String name;
    @Column(length = 20)
    @NonNull
    private String city;

    @Column(length = 20)
    @NonNull
    private String packageType;
    @NonNull
    private Double budget;
}
```

step3) add the following properties in application.properties file
for data source cfg and jpa cfgs

application.properties

```
#Datasource
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager

#Spring rest cfgs
server.port=4040
server.servlet.context-path=/RestMiniProject

# JPA cfgs
spring.jpa.database-platform=org.hibernate.dialect.OracleDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

Dec 17.1 MiniProject on Spring Rest

step4) Develop the Respository Interface extending from JpaRepository

IToursitRepo.java

```
package com.nt.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.entity.Tourist;

public interface ITouristRepo extends JpaRepository<Tourist,Integer> {

}
```

Entity class name — @Id type

The spring boot generates InMemory Impl class for our ITourist Interface having Tourist Entity class based o-r mapping persistence logic for all methods available or inherited for JpaRepository

Repo

Note:- This kind of dynamic InMemory classes will be generated in the memory where apps run nothing but JVM memory of RAM.

Note:- In JVM Memory heap for objects & statck for methods/local varibles & premgen /metaspace (from java8 metaspace is named) for InMemory code generation/ native code generation

step5) Develop Service Interface adn Service Impl class for save Object operation

Service Interface

```
package com.nt.service;
import com.nt.entity.Tourist;
public interface ITouristMgmtService {
   public  String  registerTourist(Tourist tourist);
}
```

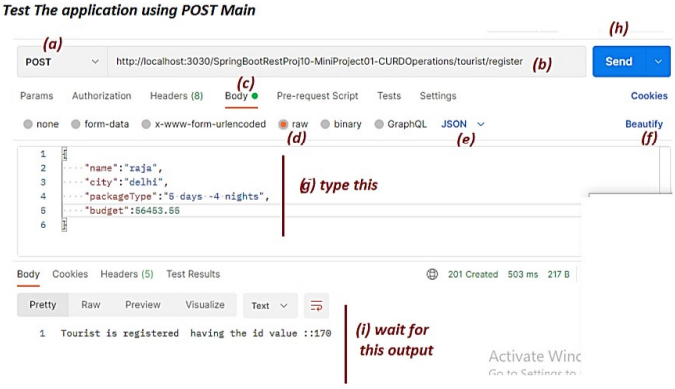
```
//service Impl class
package com.nt.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.entity.Tourist;
import com.nt.repo.ITouristRepo;
@Service("touristService")
public class TouristMgmtServiceImpl implements ITouristMgmtService {
      @Autowired
     private ITouristRepo  touristRepo;
     @Override
     public String registerTourist(Tourist tourist) {
         int idVal=touristRepo.save(tourist).getTid();
          return "Tourist is registered  having the id value ::"+idVal;
     }
}
```

step6) Develope the Restcontroller having method for save operation

```
//Controller class
ackage com.nt.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
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;
import com.nt.entity.Tourist;
import com.nt.service.ITouristMgmtService;
@RestController
@RequestMapping("/tourist")
public class TouristOperationsController {
     @Autowired
     private ITouristMgmtService  service;
     @PostMapping("/register")
     public ResponseEntity<String>  enrollTourist(@RequestBody Tourist tourist){
          try {
          //use service
          String resultMsg=service.registerTourist(tourist);
           return  new ResponseEntity<String>(resultMsg,
                                  HttpStatus.CREATED); //201 content created successfully
        }
        catch(Exception e) {
             return  new ResponseEntity<String>("problem in tourist enrollment",
         HttpStatus.INTERNAL_SERVER_ERROR); //500 error
        }
    }//method
}//class
```

step7) Run on Server application.

step8) Test The application using POST Main



Dec 18 MiniProject on Spring Rest

finalAll() operation in Mini Project

In service Interface

```
public List<Tourist> fetchAllTourists();
```

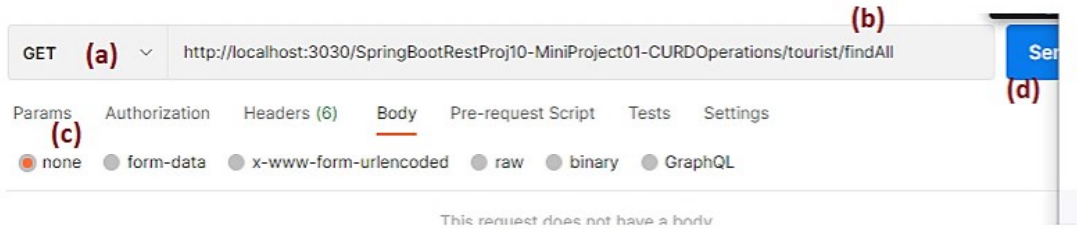
In service Impl class

```
@Override
public List<Tourist> fetchAllTourists() {
    List<Tourist> list=touristRepo.findAll();
    list.sort((t1,t2)->t1.getTid().compareTo(t2.getTid()));
    return list;
}
```

In controller class

```
@GetMapping("/findAll")
public ResponseEntity<?> displayToursits(){
    try {
        List<Tourist> list=service.fetchAllTourists();
        return new ResponseEntity<List<Tourist>>(list,HttpStatus.OK);
    }
    catch(Exception e) {
        e.printStackTrace();
        return new ResponseEntity<String>("Problem in fetching Tourists",
                HttpStatus.INTERNAL_SERVER_ERROR); //500 eror
    }
```

request from Postman tool




```
[
  {
    "tid": 170,
    "name": "raja",
    "city": "delhi",
    "packageType": "5 days -4 nights",
    "budget": 56453.55
  },
  {
    "tid": 171,
    "name": "ramesh",
    "city": "hyd",
    "packageType": "3 days -2 nights",
    "budget": 16453.55
  }
]
```

(e) get this output

Dec 18.1 MiniProject on Spring Rest

Performing findTouristByTid() operation

==========

//In service Interface

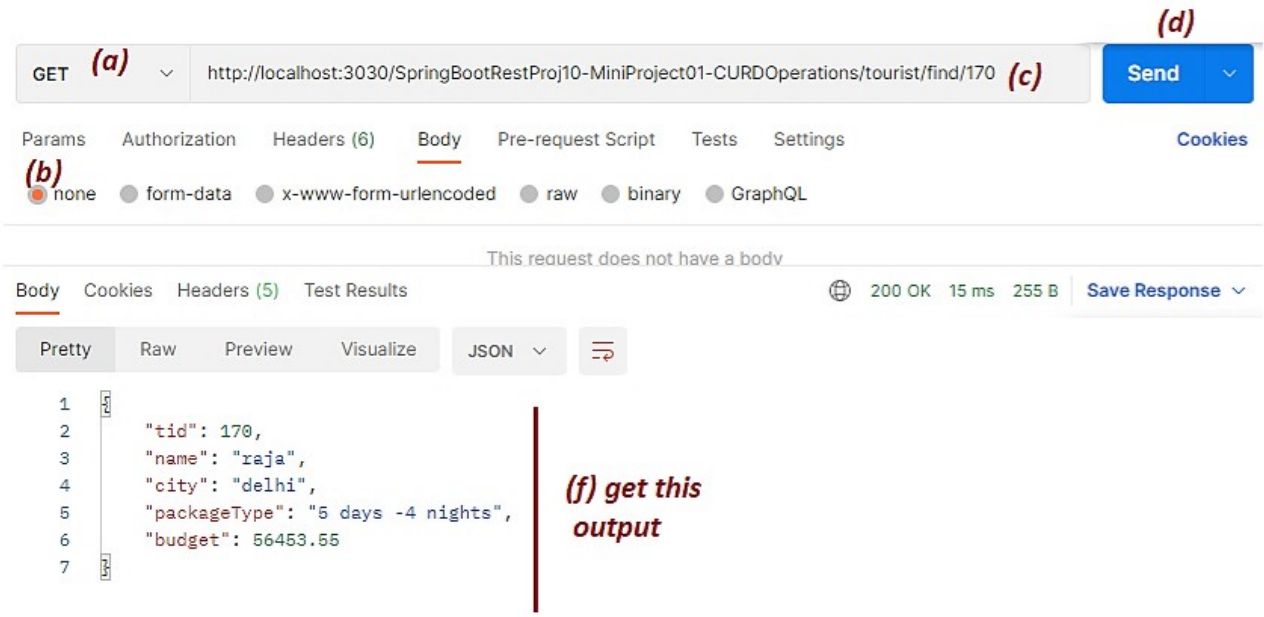
```
public Tourist fetchTouristById(Integer tid)throws TouristNotFoundException;
```

//In service impl class

```
@Override
public Tourist fetchTouristById(Integer tid) throws TouristNotFoundException {
      return touristRepo.findById(tid)
      .orElseThrow(()-> new TouristNotFoundException(tid+" tourist not found"));
}
```

In controller class

```
@GetMapping("/find/{id}")
      public ResponseEntity<?> displayTouristById(@PathVariable("id") Integer id){
            try {
                  Tourist tourist=service.fetchTouristById(id);
                  return new ResponseEntity<Tourist>(tourist,HttpStatus.OK);
            }
            catch(Exception e) {
                        e.printStackTrace();
                  return new ResponseEntity<String>(e.getMessage(),
                              HttpStatus.INTERNAL_SERVER_ERROR); //500
            }

      }//method
```



Dec 18.2 MiniProject on Spring Rest

performing update opeartion in Mini Project

In service Interface

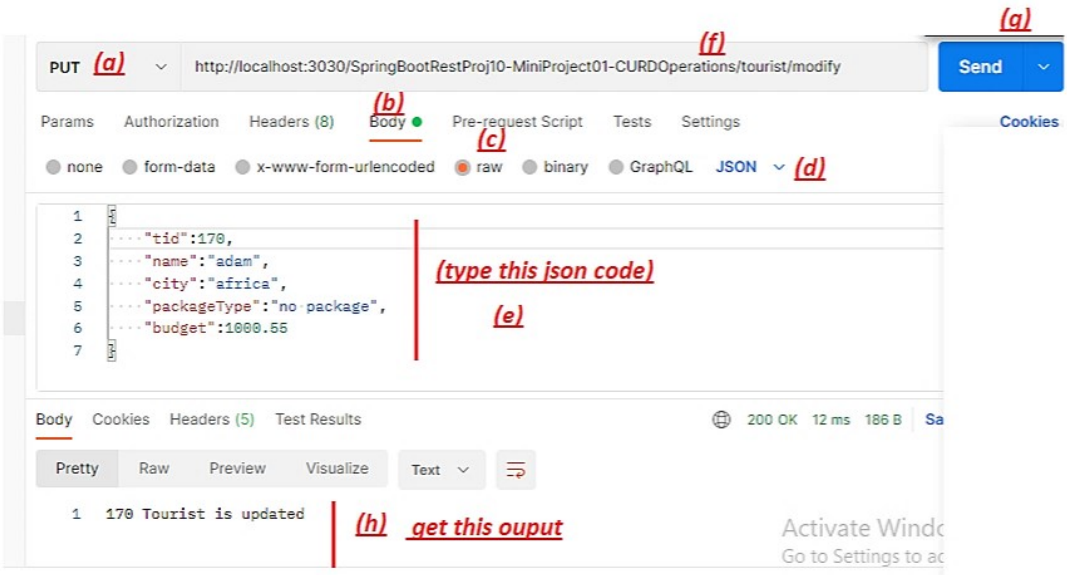
```
public   String   updateTouristDetails(Tourist tourist)throws TouristNotFoundException;
```

In service Impl class

```
@Override
public String updateTouristDetails(Tourist tourist) throws TouristNotFoundException {
      Optional<Tourist> optional=touristRepo.findById(tourist.getTid());
       if(optional.isPresent()) {
     touristRepo.save(tourist);// save(-) performs either save obj or update obj operation
     return  tourist.getTid()+" Tourist is updated";
       }
       else {
          throw new TouristNotFoundException(tourist.getTid()+" Tourist not found ");
       }

    }
```

In controller class

```
@PutMapping("/modify")
      public  ResponseEntity<String>  modifyTourist(@RequestBody Tourist tourist){
            try {
                  String msg=service.updateTouristDetails(tourist);
                  return new ResponseEntity<String>(msg,HttpStatus.OK);
            }

  catch(Exception e) {
e.printStackTrace();
return new ResponseEntity<String>(e.getMessage(),
                                  HttpStatus.INTERNAL_SERVER_ERROR);

 }
}
```



Dec 20 MiniProject on Spring Rest

Delele operation in Mini Project

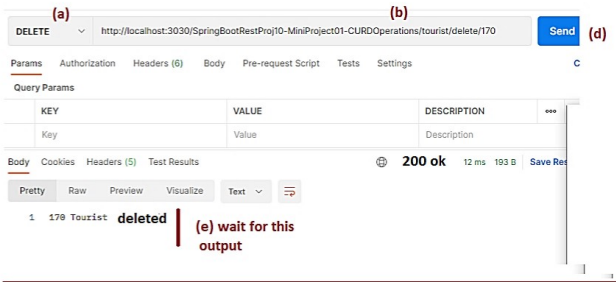
```
// In service Interface
```

```
public String deleteTourist(Integer tid)throws TouristNotFoundException;
```

```
// In service Impl class
@Override
public String deleteTourist(Integer tid) throws TouristNotFoundException {
        Optional<Tourist> opt=touristRepo.findById(tid);
        if(opt.isPresent()) {
              touristRepo.delete(opt.get());
              return tid+" Tourist deleted";
        }
        else {
           throw new TouristNotFoundException(tid+" Tourist not found ");
        }

    }
```

```
// In controller class
@DeleteMapping("/delete/{id}")
      public ResponseEntity<String> removeToursit(@PathVariable("id") Integer id){
            try {
            //use service
            String msg=service.deleteTourist(id);
             return new ResponseEntity<String>(msg,HttpStatus.OK);
            }
            catch(Exception e) {
                  e.printStackTrace();
                  return new ResponseEntity<String>(e.getMessage(),
                                                      HttpStatus.NOT_FOUND);
            }

       }//method
```

From Postman tool



Performing partial update using @PatchMapping

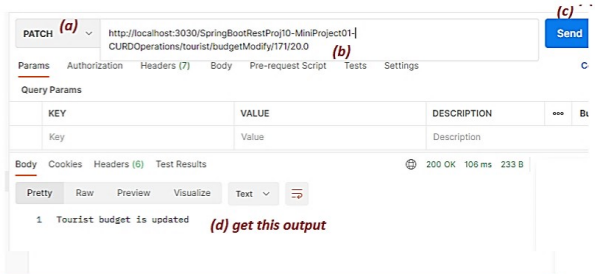
```
// In service Interface
```

```
public String updateTouristBudgetById(Integer id, Float hikePercent)throws
TouristNotFoundException;
```

```
//In service Impl class
@Override
public String updateTouristBudgetById(
                        Integer id, Float hikePercent) throws TouristNotFoundException {

   Optional<Tourist> opt=touristRepo.findById(id);
   if(opt.isPresent()) {
       Tourist tourist=opt.get();
       tourist.setBudget(tourist.getBudget()+(tourist.getBudget()*(hikePercent/100)));
       touristRepo.save(tourist);
       return "Tourist budget is updated";
   }
   else {
      throw new TouristNotFoundException(id+" Tourist not found");
   }
}
```

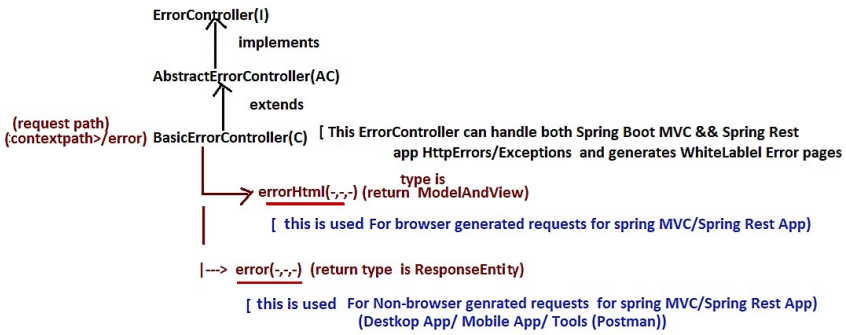
```
// In controller class
@PatchMapping("/budgetModify/{id}/{hike}")
      public ResponseEntity<String> modifyTouristBudgetById(
            @PathVariable("id") Integer id, @PathVariable("hike") Float hikePercent){

     try {
            //use service
             String msg=service.updateTouristBudgetById(id, hikePercent);
             return new ResponseEntity<String>(msg,HttpStatus.OK);
      }//try
     catch(Exception e) {
         e.printStackTrace();
         return new ResponseEntity<String>(e.getMessage(),
                                                HttpStatus.NOT_FOUND);
     }

      }
```

In Postman tool

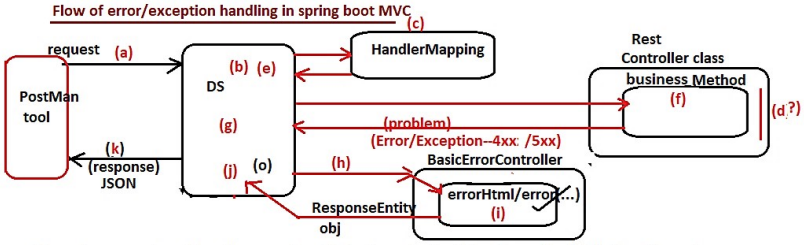


Dec 21 ErrorController-ControllerAdvice in Spring Rest

=> spring Rest/ Spring Boot MVC is having pre-defined Controller to handle these HttpErrors/exceptions and to give default whiteLabel Error pages



Flow of error/exception handling in spring boot MVC

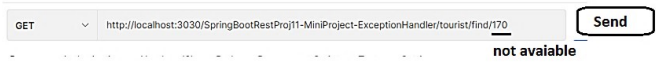


=>if we do not catch and handle exception in the b.methods of RestController (API) class then the DS gets the Propagated Exception and gives to error(-,-,-) of pre-defined controller class BasicErrorController which is mapped with <requestpath>/error (like tourist/find/error) .
=>the error(-,-,-) of BasicErrorController class returns ResponseEntity<Map<String,Object>> obj having default error messages to DS and DS converts those messages to JSON Details to send to Client as error Json Response.

To feel this pratically
==========

a) open BasicErrorController class using ctrl+shit+T option

b) get list of methods using ctrl+ o --> open error(-,-,-) source code
c) keep one breakpoint inside the error(-,-,-) method using ctrl+b or using double click in left margin
d) Run the App using Debug As server option
e) Given give request from POSTMain causing exception



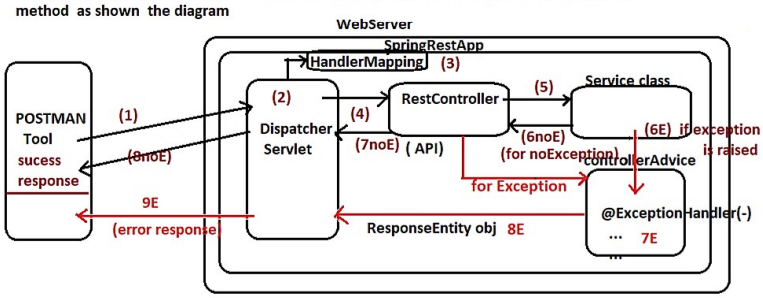
not avaiable

f) press F8 button to move control from default errorHtml(-,-,-) method to error(-,-,-) method then continuosly press f6/f7 button to go further...

g) object ResponseEntity<Map<String,Object>> object based JSON Error response in POSTMAN tool as shown below

f5 is step into
f6 step over
f7 step return
f8 next break point

=>Instead of using BasicErrorController to handle the exceptions raised or propagated to RestController methods we can use custom throws advice class that is developed using @ControlAdvice(or) @RestControllerAdvice + @ExceptionHandler which can return ResponseEntity object to DispatcherSErvlet directly by catch the exception from Service class method as shown the diagram



note:: The @ControllerAdvice or @RestControllerAdvice class @ExceptionHandler methods respond to execute for exceptions raised in Repository, service, RestController classes ..

Example App
==========

step1) Keep Mini Project ready

step2) Develop ErrorDetails class the model class to hold more details about exception

```
//ErrorDetails.java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorDetails {
      private LocalDateTime time;
      private String msg;
      private String status;

}
```

Dec 21.1 ErrorController-ControllerAdvice in Spring Rest

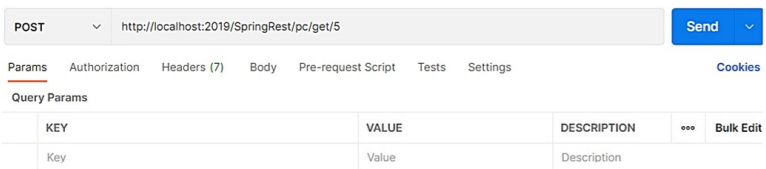
step3) Develop ControllerAdvice class as shown below

```
@ControllerAdvice
@RestControllerAdvice
public class TouristErrorHandler {

 @ExceptionHandler(TouristNotFoundException.class)
 public  ResponseEntity<ErrorDetails>  handleTouristNotFound(TouristNotFoundException tnf ){
       System.out.println("TouristErrorHandler.handleTouristNotFound()");
        ErrorDetails details=new ErrorDetails(LocalDateTime.now(),tnf.getMessage(),"404- Tourist Not Found");
        return new ResponseEntity<ErrorDetails>(details,HttpStatus.NOT_FOUND);
 }

    @ExceptionHandler(Exception.class)
    public  ResponseEntity<ErrorDetails>   handleAllProblems(Exception e){
          System.out.println("TouristErrorHandler.handleAllProblems()");
           ErrorDetails details=new ErrorDetails(LocalDateTime.now(),e.getMessage(),"Problem in execution");
           return new ResponseEntity<ErrorDetails>(details,HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

step4) Remove try catch blocks in the methods of RestController class

```
@RestController
@RequestMapping("/tourist")
public class TouristOperationsController {
      @Autowired
       private ITouristMgmtService  service;

       @PostMapping("/register")
       public ResponseEntity<String>  enrollTourist(@RequestBody Tourist tourist)throws Exception{
             //use service
             String resultMsg=service.registerTourist(tourist);
              return  new ResponseEntity<String>(resultMsg,
                                                 HttpStatus.CREATED); //201 content created successfully
       }//method
       @GetMapping("/findAll")
       public  ResponseEntity<?>  displayToursits() throws Exception{
                  List<Tourist> list=service.fetchAllTourists();
                 return new ResponseEntity<List<Tourist>>(list,HttpStatus.OK);
       }

       @GetMapping("/find/{id}")
       public  ResponseEntity<?> displayTouristById(@PathVariable("id") Integer id)throws Exception{
             System.out.println("TouristOperationsController.displayTouristById() --before");
              Tourist  tourist=service.fetchTouristById(id);
              System.out.println("TouristOperationsController.displayTouristById() --after");
                   return new ResponseEntity<Tourist>(tourist,HttpStatus.OK);

       }//method

      @PutMapping("/modify")
           public  ResponseEntity<String>  modifyTourist(@RequestBody Tourist tourist)throws Exception{
                       String msg=service.updateTouristDetails(tourist);
                       return new ResponseEntity<String>(msg,HttpStatus.OK);

           }

        @DeleteMapping("/delete/{id}")
        public  ResponseEntity<String>  removeToursit(@PathVariable("id") Integer id)throws Exception{
              //use service
             String  msg=service.deleteTourist(id);
              return  new ResponseEntity<String>(msg,HttpStatus.OK);

        }//method

        @PatchMapping("/budgetModify/{id}/{hike}")
        public  ResponseEntity<String>  modifyTouristBudgetById( @PathVariable("id") Integer id,
                                                 @PathVariable("hike") Float  hikePercent)throws Exception{
          //use service
           String msg=service.updateTouristBudgetById(id, hikePercent);
           return new ResponseEntity<String>(msg,HttpStatus.OK);
     }
    }
```

step4) Run the Application causing exception..



note : -- **if you try request to get tourist details which is not available in the db table then also it should give error response but, not in its own / same old format, it should give response having format in our own defined format those are specified details given in ErrorDetails.java" class. becoz we are explicitly created ErrorHandling class.. according to that sholud get error response also**

Same Or Old Error Response Format:--

```
{
    "timestamp": "2021-12-21T02:04:51.396+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "path": "/SpringBootRestProj11-MiniProject-ExceptionHandler/tourist/find/170"
}
```

New Error Response in Our own Format:-

```
{
    "time": "2021-12-21T21:50:08.3870323",
    "msg": "sorry the given id 5 is not valid",
    "status": "404    not found exception"
}
```

tourist

Dec 22 Swagger -Api documentation

API Creation/Development :: developing RestController having different methods/opertions for various http method types like GET,POST,PATCH,PUT,DELETE and etc... is called API Creation/Development.

=>We can generally take one @RestController per 1 module , So developing each RestController is called api creation..

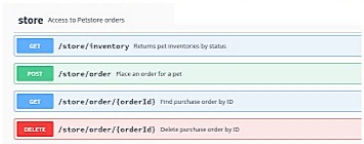
Accounts module ---> AccountsController (@RestController) is called Acccounts API creation
Daily Tx module ---> DialyTxController (@RestController) is called DailyTx API creation and etc..

EndPoints :: Providing muliple details or collection of details that are required to call methods/opertions of @RestController from Client Apps or to send reqeusts from different tools like POSTMAN is called Providing end points..

Eg:: For AccountController nothing but Accounts API the end points are

BaseURL :: http://localhost:3030/RestProj1/tourist
register() :: /register ----> POST
findById(-) :: /find/{id} ----> GET
deleteById(-) : /delete/{id} --->DELETE
and etc..

In End Points of any API we need to provide multiple details like URL ,method names, request paths , http method types ,content type and etc..



API Documentation
==========

=>we can write documentation for java classes in multiple ways
a) using serperate Text docs
b) using API documentation comments and javadoc tool

note:: Both these approaches non-responsive documentations i.e we can read about java classes and methods but we can not test them immediately

=> After developing Rest API we can provide API documentation for Rest API in the following ways

a) Using Seperate Text docs
b) Using API doc comments (/** ... */) | Creates Non-Responsive API documentation
c) Using Swagger /Swagger API (Best)
(or) Open API | Creates Responsive API Documentation.

note:: Open API is alternate to Swagger API .. The indurstry standard is still Swagger API

Swagger API
==========

=> It is an open Source Third Party Library to provide Responsive API documentation for RestController and its methods
=> For All RestControllers of the Project we can create API documentation from single place while working Swagger API
=>Responsive Documnentation means not only we get docs about API and its methods (End points) we can test immediately by providing inputs and by getting outputs..
=> if this is used .. there is no need of using POSTMAN tool seperately.. and also very useful to provide Documentation based Testing env.. from Clients.

=> Spring Fox +swagger together released libraries that are required to use swagger api in spring boot Applications

=>Swagger API documentation provides the following details in the GUI Responsive docs
a) API Info (company, title, license url , and etc..)
b) End points Info
c) Model classes info
and etc..

=> While working swagger documentation for reading and testing we need not to remember and give URL, Http method types , content type and etc.. we just need to give required inputs and get the outputs.

Procedure to work with swagger API
==========

step1) keep RestController /Rest API Project ready..

step2) Add the following two jar file in pom.xml related to swagger API

a) springfox-swagger2
a) springfox-swagger-ui

```
<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
```

Dec 22.1 Swagger -Api documentation

step3) Develope seperate Configuration class Enabling enabling Swagger api

=>In configuration class create Docket object having
->Dcoumention type (screent type)
->specify base package of restControllers
->specify reqeusts paths info
->other details of API (ApiInfo obj having company name, licenseurl and etc.)

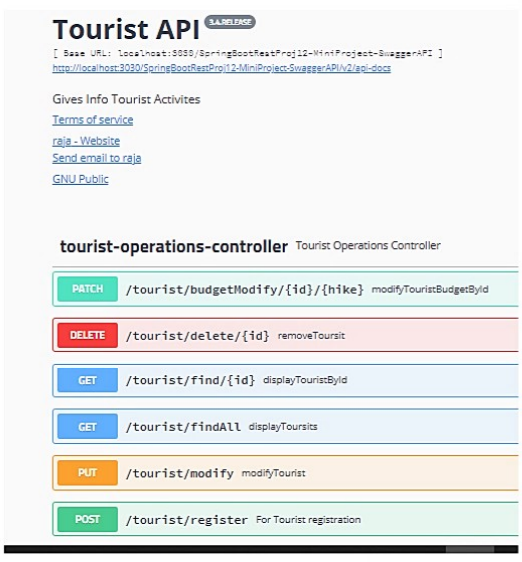
SwaggerDocConfig.java

```
package com.nt.config;

import java.util.Collections;

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

import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SwaggerDocsConfig {
@Bean
 public  Docket createDocket() {
 return  new Docket(DocumentationType.SWAGGER_2)  //UI screen type
       .select()  //to specify  RestControllers
       .apis(RequestHandlerSelectors.basePackage("com.nt.controller")) //base pkg for RestContrllers
       .paths(PathSelectors.regex("/tourist.*")) // to specify request paths
       .build() // builds the Docket obj
       .useDefaultResponseMessages(true)
       .apiInfo(getApiInfo());
 }

 private  ApiInfo getApiInfo() {
Contact contact=new Contact("raja","http://www.HCL.com/tourist","natarazjavaarena@gmail.com");
 return  new ApiInfo("Tourist API",
                     "Gives Info Tourist Activites",
                     "3.4.RELEASE",
                     "http://www.hcl.com/license",
                     contact,
                     "GNU Public",
                     "http://apache.org/license/gnu",
                     Collections.emptyList());
 }

 }
```

step4) Run the Server app

step5) use the following url to get swagger api docs and to test the api

http://localhost:3030/SpringBootRestProj12-MiniProject-SwaggerAPI/swagger-ui.html



Dec 23 RestTemplate

Developing Consumer App using RestTemplate

=> It allows to develop the consumer/Client App RestFull webService as Programmable Client App in java env..

=> We need to take seperate WebService/MVC Project for this having logics to consume webService /API by calling methods.

=> This object (RestTemplate) does not come through AutoConfiguration Process .. It must be created either using "new" operator or using @Bean method

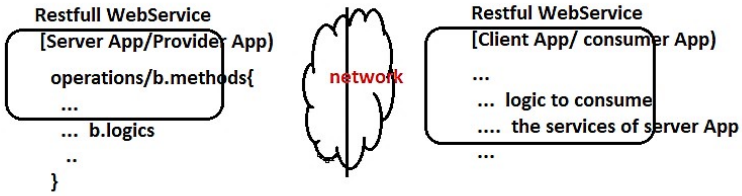
```
RestTemplate template=new RestTemplate();
        (or)
In @Configuration class
----------------------------------
  @Bean("template")
  public  RestTemplate createTemplate(){
        return new  RestTemplate();
   }
```

=> This object provide methods to generate different modes requests like GET/POST/PUT/DELETE/.... to consume the Restfull webService /API.. i.e we can call methods /operations Restful webService Server App/provider App

=> While using this object to cosume RestFull WebService/API we need detailed inputs (nothing but end points) like base url , http method type, http header info like content type and etc..

=> It provides xxxForEntity(....) methods like getForEntity(...) , postForEntity(...) and etc.. taking url, request obj(body,header) to send different modes http requests as method calls to cosume the the Restfull web service (Server/Provider App)

=> Do not forget WebService is given to link two different Apps that are developed either in same language or in diferent languages and in running same server or different servers belonging to same machine or different machines.



Server/Provider App — Client /Cosumer App

PaymentGateway <----------> PayTM App

PayTM/GooglePay/PhonePay<----------> Flipkart/Amazon

WeatherReport App <----------> yahoo.com/Tourist.com

ICC Score Comp <----------> CrickInfo.com, CrickBuzz.com and etc..

=>The RestFullWebService (Server/ provider App) must be the web application

=> The consumer App can be the standalone App or mobile App or IOT App or Web application or etc..

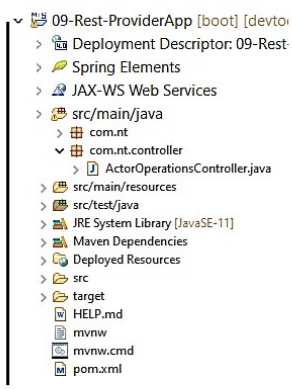
So far we have devleop only RestFul WebService(Server/provider App) and we tested that server App using tools like POSTMAN/Swagger ... Instread of using these tools we can develop programmable Real client Apps with the support RestTemplate in spring Env..

=>RestTemplate can be used only in Spring or Spring Boot env..

Example App

step1) Develop Restful WebService App as web application (Server /provider App)
(old style App)

staters :: web, dev tools, lombok api



step2) Develop API/Rest controller

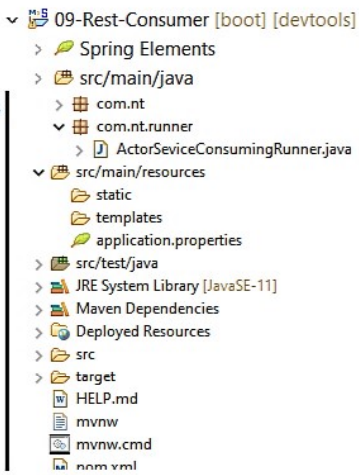
```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/actor")
public class ActorOperationsController {

   @GetMapping("/wish")
   public  ResponseEntity<String>  displayWishMessage(){
         return  new ResponseEntity<String>("Good Morning",HttpStatus.OK);
   }
  }
```

step3) Run The application...on server (Run As --->Run on SErver)

Dec 23.1 RestTemplate

Process-1: run both Server(Provider) & Consumer(Client) Apps on External Server

step4) Develop the Consumer App as seperte Project
(starters :: web , dev tools , lombok)



step5) place the following entries in application. properties
server.port=4040

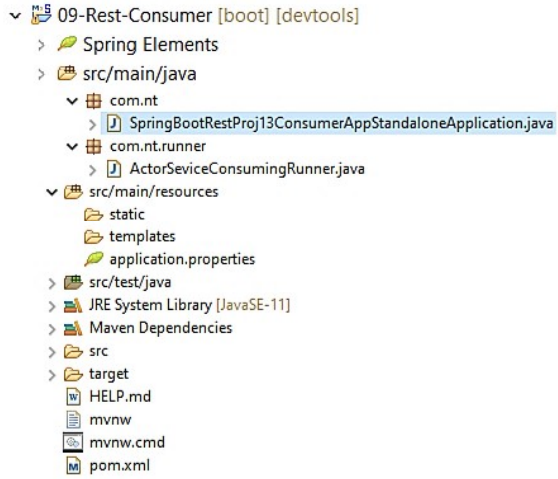
step6) Develop the Runner App

```
package com.nt.runner;

import org.springframework.boot.CommandLineRunner;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class ActorSeviceConsumingRunner implements CommandLineRunner {

@Override
public void run(String... args) throws Exception {
      //create RestTemplate class object
      RestTemplate  template=new RestTemplate();
      //Define  service url
       String serviceUrl="http://localhost:3030/SpringBootRestProj13-ProviderApp/actor/wish";
       // Generate Http reqeust   with GET mode to consume the web service(API)
       ResponseEntity<String> response=template.getForEntity(serviceUrl, String.class);
       //display the recieved details from the response
       System.out.println("Response body(output) ::"+response.getBody());
       System.out.println("Response status code value ::"+response.getStatusCodeValue());
       System.out.println("Response satus code ::"+response.getStatusCode().name());

       //System.exit(0); //optional

    }
 }
```

Process-2: Run Provider on Server & Consumer on Embedded Server(as Boot App)

step4) Run Consumer App as spring boot App that uses Embedeed Tomcat server..
Run As ----> spring Boot App/ jav aApp

note:: The above consumer App can also be develop as standalone app (package type is jar)
adding spriing web starters.. as shown below



Runner class code
and application.propeeties
content is same as the above
Consumer App..

Dec 24 RestTemplate

What is the difference b/w getforEntity(..) and getForObject(..) methods?

Ans) getForEntity(..) return type is ResponseEntity<T> which contains response body(output), headers, status code and etc.. (Best Approach)

getForObject(..) return type is Object which gives only response body(output) i.e it does not give response headers, status code and etc..

note:: when we develop spring web mvc/spring rest app as war file(web application) we can use both extenal server and Embedded Server (like Embedded Tomcat) for deployment.

note:: when we develop spring web mvc/spring rest app as jar file(standalone web application) we can use only Embedded server (like Embedded Tomcat) for deployment.

Passing Path variable values along with http request calls using RestTemplate :-

=>Both getForObject(...), getForEntity(...) are having multiple overloaded forms .. the form that is taking more params is maintaining last param as var args param which is given to pass any no.of path variable values along with http request call.

`<T> T`	`getForObject(String url, Class<T> responseType, Map<String,?> uriVariables)` Retrieve a representation by doing a GET on the URI template.	To pass path variable names and vlaues as map collection	Need not to fallow the order while passing values
`<T> T`	`getForObject(String url, Class<T> responseType, Object... uriVariables)` Retrieve a representation by doing a GET on the specified URL.	To pass path variable vlaues as var args..	Needs to fallow the order while passing values

`<T> ResponseEntity<T>`	`getForEntity(String url, Class<T> responseType, Map<String,?> uriVariables)` Retrieve a representation by doing a GET on the URI template.	To pass path variable names and vlaues as map collection	Need not to fallow the order while passing values
`<T> ResponseEntity<T>`	`getForEntity(String url, Class<T> responseType, Object... uriVariables)` Retrieve an entity by doing a GET on the specified URL.	To pass path variable vlaues as var args..	Needs to fallow the order while passing values

Example App

In server /producer / Provider App

RestControlller

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {

    @GetMapping("/wish/{id}/{name}")
    public ResponseEntity<String> displayWishMessage(@PathVariable Integer id,
                                                     @PathVariable String name){
        return new ResponseEntity<String>("Good Morning::"+id+"..."+name,HttpStatus.OK);
    }

}
```

In Consumer App/Client App /

runner class

```
@Component
public class ActorSeviceConsumingRunner_PathVariables implements CommandLineRunner {

@Override
public void run(String... args) throws Exception {
    //create RestTemplate class object
    RestTemplate template=new RestTemplate();
    //Define service url
    String serviceUrl="http://localhost:3030/SpringBootRestProj13-ProviderApp/actor/wish/{id}/{name}";

    // Generate Http reqeust with GET mode to consume the web service(API)
    //ResponseEntity<String> response=template.getForEntity(serviceUrl, String.class,1001,"raja"); (or)
    ResponseEntity<String> response=template.getForEntity(serviceUrl, String.class,Map.of("name","rajesh","id",1001));

    //display the recieved details from the response
    System.out.println("Response body(output) ::"+response.getBody());
    System.out.println("Response status code value ::"+response.getStatusCodeValue());
    System.out.println("Response headers ::"+response.getHeaders().toString());
    System.out.println("Response satus code ::"+response.getStatusCode().name());


    System.exit(0); //optional .. given for to stop server automatically

}

}
```

Dec 24.1 RestTemplate

Sending JSON Data from Consumer app along with POST mode request using RestTemplate

=> POST mode request contains request body, request headers and initial line
where as GET mode request contains only request headers and initial line.

Every request sturcture contains 2 parts
HEAD part (initial line + request headers)
BODY/Payload part (request body)

HEAD, BODY of request structure will be seperate with blankline

=> Here we need to use postForEntity(..) or postForObject(...) methods of RestTemplate to send post mode request and to get response completelty or partially..

`<T> ResponseEntity<T>`	`postForEntity(String url, Object request, Class<T> responseType, Map<String,?> uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the response as `HttpEntity`.
`<T> ResponseEntity<T>`	`postForEntity(String url, Object request, Class<T> responseType, Object... uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the response as `ResponseEntity`.
`<T> ResponseEntity<T>`	`postForEntity(URI url, Object request, Class<T> responseType)` Create a new resource by POSTing the given object to the URL, and returns the response as `ResponseEntity`.

`<T> T`	`postForObject(String url, Object request, Class<T> responseType, Map<String,?> uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the representation found in the response.
`<T> T`	`postForObject(String url, Object request, Class<T> responseType, Object... uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the representation found in the response.
`<T> T`	`postForObject(URI url, Object request, Class<T> responseType)` Create a new resource by POSTing the given object to the URL, and returns the representation found in the response.

In server/ Producer/Provider App

Create a model class

```
package ss.it.model;
import lombok.Data;
@Data
public class Actress {
      private Integer aid;
      private String name;
      private Float age;
      private String type;
}
```

RestController

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {


    @PostMapping("/register")
    public ResponseEntity<String> registerActor(@RequestBody Actor actor){
          return new ResponseEntity<String>("Actor data"+actor.toString(),HttpStatus.OK);
    }
}
```

In Consumer App/Client App

Runner class

Note:- Setting media type is optional for JSON, Text. only for XML we use it

```
@Component
public class ActorSeviceConsumingRunner_PostingJSON_Data implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
          //create RestTemplate class object
          RestTemplate template=new RestTemplate();
          //Define service url
          String serviceUrl="http://localhost:3030/SpringBootRestProj13-ProviderApp/actor/register";
          //prepare JSON data (request body)
          String json_body="{ \"aid\" : 1001, \"name\": \"suresh\" ,\"age\": 30.0 ,\"type\":\"hero\" }";
          //prepare headers
          HttpHeaders headers= new HttpHeaders();
          headers.setContentType(MediaType.APPLICATION_JSON);
          // prepare Http request as HttpEntity obj having head , body
          HttpEntity<String> request= new HttpEntity<String>(json_body,headers);
          // make http request call in post mode
      ResponseEntity<String> response=template.postForEntity(serviceUrl,request,String.class); //url,request, output type

          //display the recieved details from the response
          System.out.println("Response body(output) ::"+response.getBody());
          System.out.println("Response headers ::"+response.getHeaders().toString());
          System.out.println("Response status code value ::"+response.getStatusCodeValue());
          System.out.println("Response satus code ::"+response.getStatusCode().name());


          System.exit(0); //optional .. given for to stop server automatically

    }
}
```

note:-

plese check the json data before running the application if you type wrongly you may get 400 error

if you getting the problem while perparing json data as request body follow the steps::::

step 1:- put " ";
step 2:- put "{ }";
step 3:- put "{ key : value }";
final step :- put this symbols at the at starting of key and value, ending of key and value

Dec 26 RestTemplate exchange(-) method

Working with RestTemplate

=>Instead of calling getForXxx() ,postForXxx() , put(..),delete() and etc.. as seperate methods to generate different modes of requests .. we can use single exchange(....) for all operations. (one method to generate all modes of requets)

```
public <T> ResponseEntity<T> exchange(String url,
                                      HttpMethod method,
                                      @Nullable
                                      HttpEntity<?> requestEntity,
                                      Class<T> responseType,
                                      Object... uriVariables)
                               throws RestClientException
```

Execute the HTTP method to the given URI template, writing the given request entity to the request, and returns the response as ResponseEntity.

URI Template variables are expanded using the given URI variables, if any.

Specified by:
exchange in interface RestOperations

Parameters:
url - the URL
method - the HTTP method (GET, POST, etc)
requestEntity - the entity (headers and/or body) to write to the request may be null) **(headers+ body)**
responseType - the type to convert the response to, or Void.class for no body **(required response type)**
uriVariables - the variables to expand in the template **(path variables values)**

Returns:
the response as entity

Throws:
RestClientException

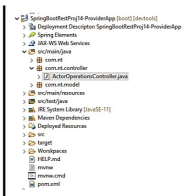
exchnage(.....) is alternate for the following methods

getForEntity(...) , postForEntity(...) , getForObject(...) , postForObject(...), delete(...) ,put(...) , patchForEntity(...) , patchForObject(...) and etc..

Example app

Provider/Producer/Server App

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {
      @GetMapping("/wish")
      public ResponseEntity<String> displayWishMessage(){
            return new ResponseEntity<String>("Good Morning",HttpStatus.OK);
      }
      @GetMapping("/wish/{id}/{name}")
      public ResponseEntity<String> displayWishMessage(@PathVariable Integer id,
                                                        @PathVariable String name){
            return new ResponseEntity<String>("Good Morning::"+id+"..."+name,HttpStatus.OK);
      }
      @PostMapping("/register")
      public ResponseEntity<String> registerActor(@RequestBody Actor actor){
            return new ResponseEntity<String>("Actor data"+actor.toString(),HttpStatus.OK);
      }
}
```



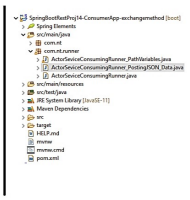
Use Run on server .....

Client App/ consumer App

```
@Component
public class ActorSeviceConsumingRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
 System.out.println("ActorSeviceConsumingRunner.run()");
 //create RestTemplate class object
 RestTemplate  template=new RestTemplate();
 //Define  service url
  String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/wish";

  //  invoke  service method/operation using exchange(-,-,-) method
  ResponseEntity<String>  resp=template.exchange(
                                serviceUrl,
                                HttpMethod.GET,
                                null,  // no  body for GET moode request and
                                       // we do not want to pass  any header values
                                String.class);

//  display the details
System.out.println("response body (output)::"+resp.getBody());
System.out.println("response status code ::"+resp.getStatusCode());
System.out.println("response status code value ::"+resp.getStatusCodeValue());
System.out.println("response header values ::"+resp.getHeaders());


System.out.println("-----------------------------------------");
    }
}
```

```
@Component
public class ActorSeviceConsumingRunner_PathVariables implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
 //create RestTemplate class object
RestTemplate  template=new RestTemplate();
//Define  service url
 String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/wish/{id}/{name}";

 //  invoke  service method/operation using exchange(-,-,-) method
 ResponseEntity<String>  resp=template.exchange(
                               serviceUrl,
                               HttpMethod.GET,
                               null,  // no  body for GET moode request and
                                      // we do not want to pass any header values
                               String.class,  //required type
                               7890, "anitha");  //path or uri variable values

 //  display the details
 System.out.println("response body (output)::"+resp.getBody());
 System.out.println("response status code ::"+resp.getStatusCode());
 System.out.println("response status code value ::"+resp.getStatusCodeValue());
 System.out.println("response header values ::"+resp.getHeaders());

System.out.println("-----------------------------------------");
    }
}
```

```
@Component
public class ActorSeviceConsumingRunner_PostingJSON_Data implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
          //create RestTemplate class object
          RestTemplate  template=new RestTemplate();
          //Define  service url
           String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/register";
           //prepare JSON data  (request body)
           String json_body="{ \"aid\" : 1002, \"name\": \"ranveer\" ,\"age\": 30.0 ,\"type\":\"hero\" }";
           //prepare headers
           HttpHeaders headers= new HttpHeaders();
           headers.setContentType(MediaType.APPLICATION_JSON);
           // prepare  Http request as HttpEntity obj having head , body
           HttpEntity<String> entity= new HttpEntity<String>(json_body,headers);  //headers + body
           // make http request call  in post mode
     ResponseEntity<String>  response=template.exchange(serviceUrl,
                                                        HttpMethod.POST,
                                                        entity,
                                                        String.class) ;

          //display the recieved details from the response
          System.out.println("Response body(output) ::"+response.getBody());
          System.out.println("Response headers ::"+response.getHeaders().toString());
          System.out.println("Response status code value ::"+response.getStatusCodeValue());
          System.out.println("Response satus code ::"+response.getStatusCode().name());

          System.out.println("-----------------------------------------");
    }
}
```



note: Instead of creating Multiple objects for RestTemplate class in different runner classes of Cosumer App .. create RestTemplate class obj in main class using @Bean method and inject the spring bean obj in runner classes with the support @Autowired annotation.

Recieving and processing diferent return type values to webService methods/operations using RestTemplate

=> If web service method /operation return type is ResponseEntity<Strinng> /String then we get plain text as the response..

=> If web service method /operation return type is othe than ResponseEntity<String> /String then we get JSON response as text content.. by default .. this JSON text content can be used directly or can be converted to Java object using JACKSON API (built -in spring boot Rest Applications)

=>Once we add spring MVC starter to the Project ... we get JACSON API automatically we can be used to JSON data to Object (DeSerilization) and Object to JSON data (Serialization)

=> Irrespective of the return value of web service method/operation (String or non-String) we generally take the return value of exchnage(...) or getForXxx(...) or postForXxx(...) and etc.. methods as String based return value like ResponseEntity<String> object.. i.e recieve every thing from provider App as String content and convert it to other types if needed.

=>The ObjectMapper class of Jackson api gives methods for Serilziation and DeSerilizatation

a) mapper.readValue(-) :: JSON text to Object (DeSerialization)
b) mapper.writeValue(-) :: Object to JSON text (Serialization)

Sever App/proividerApp/Producer App

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {
      @GetMapping("/find")
      public  ResponseEntity<Actor>  searchActor( ){
            return new ResponseEntity<Actor>(new Actor(200,"salman",56.0f,"hero"),  //body
                                             HttpStatus.OK);  // status code

      }
}
```

Client App/Consumer App

```
@Component
public class ActorSeviceConsuming_GettingJSONData_Runner implements CommandLineRunner {
      @Override
      public void run(String... args) throws Exception {
            System.out.println("ActorSeviceConsuming_GettingJSONData_Runner.run()");
            //create RestTemplate class object
            RestTemplate  template=new RestTemplate();
            //Define  service url
             String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/find";

             // invoke  service method/operation using exchange(-,-,-) method
             ResponseEntity<String>  resp=template.exchange(serviceUrl,
                                                            HttpMethod.GET,
                                                            null,  // no  body for GET moode request and  we do not want to pass
                                                                   //any header values
                                                            String.class);
             //  display the details
             System.out.println("response body (json output)::"+resp.getBody());
             System.out.println("response status code ::"+resp.getStatusCode());
             System.out.println("response status code value ::"+resp.getStatusCodeValue());
             System.out.println("response header values ::"+resp.getHeaders());
             System.out.println("-----------------------------------------");
             //converting JSON text response(body) to  Java class object /Model class obj/Entity class object using JACKSON api
              String jsonBody=resp.getBody();
              //create ObjectMapper
              ObjectMapper mapper=new ObjectMapper();
              Actor actor=mapper.readValue(jsonBody,  // body
                                           Actor.class);  //requred object type
              System.out.println("response body as the Actor object data:: "+actor);

      }//method
}//class
```

Dec 28 RestTemplate all methods

Converting List of objects Data (1D Array Data) of JSON given By Server/Producer/Provider App into List of Objects in Conumer App using Jackson api

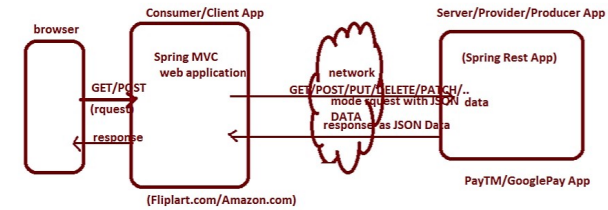
In server /Producer/Provider App

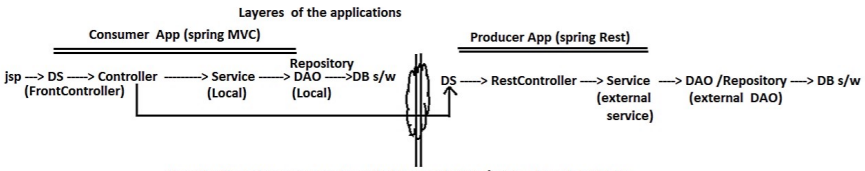
```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {

   @GetMapping("/findAll")
        public  ResponseEntity<List<Actor>>  fetchAllActors(){
              return new  ResponseEntity<List<Actor>>(List.of(new Actor(101,"salman",55.0f,"hero"),
                                                              new Actor(102,"rajesh",65.0f,"hero"),
                                                              new Actor(103,"ranveer",35.0f,"hero")),
                                                             HttpStatus.OK);
        }
 }
```

In Consumer/Client App

```
@Component
public class ActorSeviceConsuming_GettingJSONData_Runner implements CommandLineRunner {

@Override
public void run(String... args) throws Exception {
 System.out.println("ActorSeviceConsuming_GettingJSONData_Runner.run()");
 //create RestTemplate class object
 RestTemplate  template=new RestTemplate();

 //Define  service url
 String    serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/findAll";
 //  invoke  service method/operation using exchange(-,-,-) method
       ResponseEntity<String>  resp1=template.exchange(serviceUrl,
                                    HttpMethod.GET,
                                    null,  // no  body for GET moode request and  we do not want to pass
                                           //any header values
                                    String.class);

//  display the details
System.out.println("response body (json output)::"+resp1.getBody());
System.out.println("response status code ::"+resp1.getStatusCode());
System.out.println("response status code value ::"+resp1.getStatusCodeValue());
System.out.println("response header values ::"+resp1.getHeaders());
//converting JSON text response(body) to  Java class object /Model class obj/Entity class object using JACKSON api
 String jsonBody1=resp1.getBody();
 //create ObjectMapper
 ObjectMapper mapper1=new ObjectMapper();
 Actor[] actors=mapper1.readValue(jsonBody1,   // body
                                  Actor[].class);  //required object type
 List<Actor> listActors=Arrays.asList(actors);
 System.out.println("response body as the List<Actor> object's data:: "+listActors);
 System.out.println("-----------------------------");
 listActors.forEach(System.out::println);
 System.out.println("-----------------------------");
 List<Actor> listActors1=mapper1.readValue(jsonBody1,new TypeReference<List<Actor>>() {});
 System.out.println("-----------------------------");
 listActors1.forEach(System.out::println);

    }//method
}//class
```

TypeReference is jackson api supplied
abstract class is used for obtaining full generics type information by sub-classing;





The Distributed comps are also called External comp/External services becoz they will not part of Consumer App/Project .. They always reside outside the of consumer Project/App giving services to Consumer App.

what is the difference b/w put(-) and exchange(-,-,-,-) of RestTemplate?

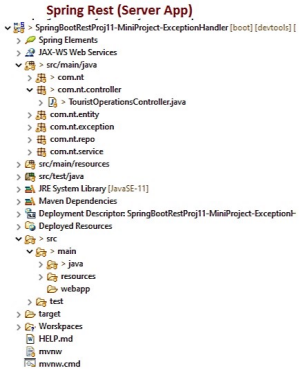
Ans) =>put(-) mehthod can send only PUT mode request producer App where as exchange(...) can send different modes of requests

=>put(-) mehthod return type is void i.e we can not get response body given by producer App where as exchange(...) return type is ResponseEntity<T> i.e we can get result/response given by provider App

what is the difference b/w delete :(-) and exchange(-,-,-,-) of RestTemplate?

Ans) Similar to above

Spring Rest (Server App)



Spring MVC App (Cosumer App)