web service application is developed by using certain standard rules.

if web app follows SOAP ( Simple Object Access Protocol ) protocol then it is called SOAP web services.

if web app follows REST guidelines then it is called REST web services.

the file which contains all basic information about the specific web

service application is known as WSDL (Web Service Description Language)

client and web service application or service provider communicate each

other by using SOAP format or SOAP protocol is called SOAP web service.

SOAP Protocol says communication between two softwares should happen in XML format

by following certain rules.SOAP web services and its client use other protocols instead of

http like SMTP, FTP, JMS etc as per the requirement.

client and web service application or service provider communicate each

other by using REST guidelines then it is called REST web service.

**REST** - Representation State Transfer. It is basically a set of guidelines which describes

the overall architecture of the application.

In architecture of client and server the

Client requests for a resource to the server and server gives the response in the form of

Representation of resource+additional info

Representation of resource is nothing but copy of requested resource in any one of formats like xml,

json, html, csv, pdf etc and it includes the neccessary links and uri which may be helpful to

the client.

additional info consists of charactertistic of representation of resource like format, size etc.

jackson-dataformat-xml beacuse of this spring is giving output in the form of xml else it will be in jason.

**@ResponseBody** returns the java object but if jackson jars are there then java object will be converted into

json type.

(com.fasterxml.jackson.annotation.JsonProperty;)

**@JsonProperty**("Student\_Name") - used for show the user defined field name.

**@JsonProperty**("Student Name") :-

This page contains the following errors:

error on line 1 at column 47: Specification mandates value for attribute Name

Below is a rendering of the page up to the first error.

**@JsonPropertyOrder**({"regNo","name", "cgpa"}) - sets the order of the fields.

**@JsonIgnoreProperties**({"cgpa"}) - it doesn't include the cgpa field.

**@JsonInclude**(JsonInclude.Include.NON\_NULL) - will exclude the null values.

\*\*Repeated annotations are allowed on 1.8 source and above.....

**@JsonInclude**(JsonInclude.Include.NON\_DEFAULT) - ignores default values

// retrieves Single object from the database.....

@ResponseBody

@RequestMapping(value = "/students/{name}", method = RequestMethod.GET)

public student getStudent(@PathVariable("name") String name)

{

student stud = new student();

stud.setName(name);

stud.setCgpa(8.56);

return stud;

}

**@RestController** - improves the readability of the REST api or spring MVC project.

whatever the mehods inside the restcontroller are basically related to rest api

so for these methods we don't have to provide the @ResponseBody explicitly

Sophsticated REST API testing tools are :

1.SOAP UI

2.RESTlet Client

3.POSTMAN

In POSTMAN testing tool we use headers in that key as Accept and value as application/json

or application/xml is used to convert the response into json form xml form.

The scenario where the clients can read the data which is available on the server then we

have to use Http GET request.

The scenario where the clients need to update the information which kept at the server side

then we go for PUT request.

In order to request to be successful clients sents some info either in format of json or xml

to the server.

@RequestBody is responsible to convert json or xml into equivalent java object.

JSON:-

{

"StudentName":"pradeep",

"cgpa":8.5

}

XML:-

<student>

<StudentName>Lakshman</StudentName>

<cgpa>8.6</cgpa>

</student>

If we want to restrict the Rest API controller method to support only XML format then

we have to use the **consumes** argument in the @RequestMapping annotation.

@RequestMapping(value = "/students/{name}", method = RequestMethod.PUT, consumes = MediaType.APPLICATION\_XML\_VALUE)

public boolean method(@PathVariable("name") String name, @RequestBody student stud)

{

System.out.println("Student : "+name);

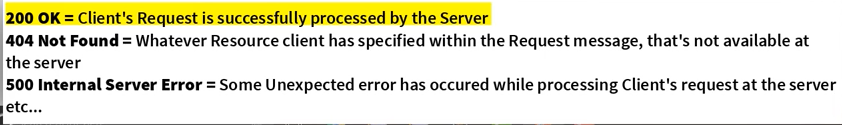
System.out.println("Student Name : "+stud.getName());

System.out.println("Student CGPA : "+stud.getCgpa());

return true;

}

If we change value as application/json on postman then spring will throw an error : org.springframework.http.converter.HttpMessageNotReadableException: Could not read document:



If method is not throwing any exception then application by default it sets as status ok :200

**ResponseEntity:** It is used to set status of the method. If we are using it as a return type then that controller method to be able to specify response body and status within it.

@RequestMapping(value = "/students/{name}", method = RequestMethod.***PUT***, consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<Void> method(@PathVariable("name") String name, @RequestBody student stud)

{

System.***out***.println("Student : "+name);

System.***out***.println("Student Name : "+stud.getName());

System.***out***.println("Student CGPA : "+stud.getCgpa());

// return true;

**return** **new** ResponseEntity<Void>(HttpStatus.***OK***);

}

If we want to send the response body also to the application then refer the code below, here in ResponseEntity<Boolean>(**true**, HttpStatus.***OK***); first argument refers to the response body and second refers to the response status.

Here ResponseEntity type depends on the type of the Response body.

@RequestMapping(value = "/students/{name}", method = RequestMethod.***PUT***, consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<Boolean> method(@PathVariable("name") String name, @RequestBody student stud)

{

System.***out***.println("Student : "+name);

System.***out***.println("Student Name : "+stud.getName());

System.***out***.println("Student CGPA : "+stud.getCgpa());

// return true;

**return** **new** ResponseEntity<Boolean>(**true**, HttpStatus.***OK***);

}

**Http Response Headers :** Headers in the postman contains key value pairs which describes the overall response message in a more appropriate manner.

**content-type →**application/xml;charset=UTF-8 - it tells to client that in what format response body come to it.

**date →**Thu, 06 Apr 2023 07:25:09 GMT - it tells at what date and time this response was produced by the server

**server →**Apache-Coyote/1.1 - it tells what kind of web server has produced this response

**transfer-encoding →**chunked - it tells about what kind of encoding technique has been used to transfer this response from server to the client.

@RequestMapping(value = "/students/{name}", method = RequestMethod.***PUT***, consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<Boolean> method(@PathVariable("name") String name, @RequestBody student stud)

{

System.***out***.println("Student : "+name);

System.***out***.println("Student Name : "+stud.getName());

System.***out***.println("Student CGPA : "+stud.getCgpa());

org.springframework.http.HttpHeaders head = **new** org.springframework.http.HttpHeaders();

head.add("Name", "Lakshman");

head.add("RegNo", "12009064");

**return** **new** ResponseEntity<Boolean>(**true**, head, HttpStatus.***OK***); }

**POST REST API:-** We use this for creating new resources in the server side.

The developers will send the request status as 201 created for POST RequestMethod.

The status 200 OK signifies whatever request that client had made to the server that request is successfully processed. The status 201 created signifies that whatever resource client wanted to create at the server that successfully created.

@RequestMapping(value = "/students", method = RequestMethod.***POST***, consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<Boolean> method2(@RequestBody student stud)

{

System.***out***.println("Student Name : "+stud.getName());

System.***out***.println("Student CGPA : "+stud.getCgpa());

// return true;

// HttpHeaders head = new HttpHeaders();

// head.add("Name", "Lakshman");

// head.add("RegNo", "12009064");

// return new ResponseEntity<Boolean>(true, head, HttpStatus.OK);

HttpHeaders head = **new** HttpHeaders();

head.add("Location", ServletUriComponentsBuilder.*fromCurrentRequest*().path("/{name}").buildAndExpand(stud.getName()).toUri().toString());

**return** **new** ResponseEntity<Boolean>(**true**, head, HttpStatus.***CREATED***);

}

**DELETE REST API:-**

@RequestMapping(value = "/students/{name}", method = RequestMethod.***DELETE***)

**public** ResponseEntity<Boolean> method(@PathVariable("name") String name)

{

System.***out***.println("Student : "+name);

// System.out.println("Student Name : "+stud.getName());

// System.out.println("Student CGPA : "+stud.getCgpa());

// return true;

// HttpHeaders head = new HttpHeaders();

// head.add("Name", "Lakshman");

// head.add("RegNo", "12009064");

**return** **new** ResponseEntity<Boolean>(**true**, HttpStatus.***OK***);

}

**HATEOAS –** Hypermedia As The Engine Of Application State

**Hypertext –** It is a text that has the links to the other text.

**REL –** The REL attribute specifies the relationship between the current document and the link document. Or It adds extra information in the links in our rest response and it is a part of the HTTP specification.

HATEOAS is the way to provide links to resources in the API response so that the client doesn’t have to deal with the URI.