206: Introduction to Rest Service:

In real world, there might be scenarios where multiple applications they want to communicate with each other are one application, want to connect with the other application, which are hosted in completely different servers.

So for these kind of communication requirements or for these kind of interactions between multiple applications which are hosted in multiple servers, we need web services.

Web services is a development pattern using which multiple applications they can interact with each other regardless of what is the underlying technology.

We can build a web service based upon the Java language and and I can expose my web service through an endpoint URL. So other developer who is building an another application with another technology like .net, Python or Mainframe.So regardless of whatever technology or language that they are using, they can make a HTTP request

to my web service endpoint URL, and that way they can always communicate with my web application and based upon the agreement that we have between these two applications, the data will be exchanged from one application to other application.

We can build web service in two styles:

SOAP --Simple Object Access Protocol.

Soap based web services has pre-defined schema and they strictly follow XML Style of Communication.

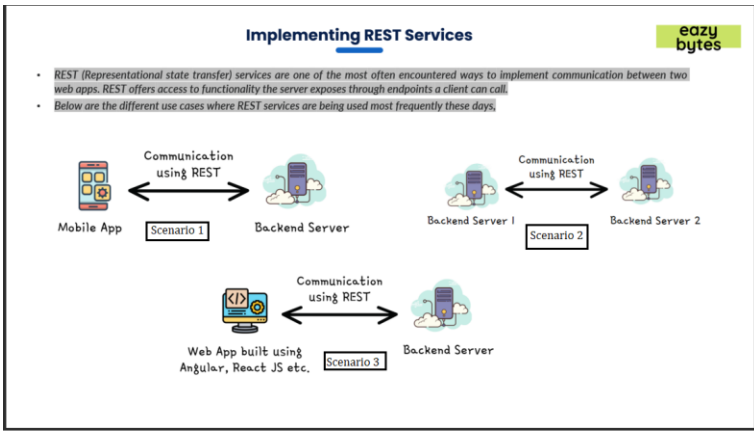
Since Soap based services are heavy in nature every developer is using Rest based web Services.

Rest based Web Services:

In case of Rest based Web Services we don't need a pre-defined schema or pre-defined contract between the applications.

Other Advantage is in case of Rest based Web Services Communication can happen in the format of XML or JSON.

JSON is light weight in nature for building web services.

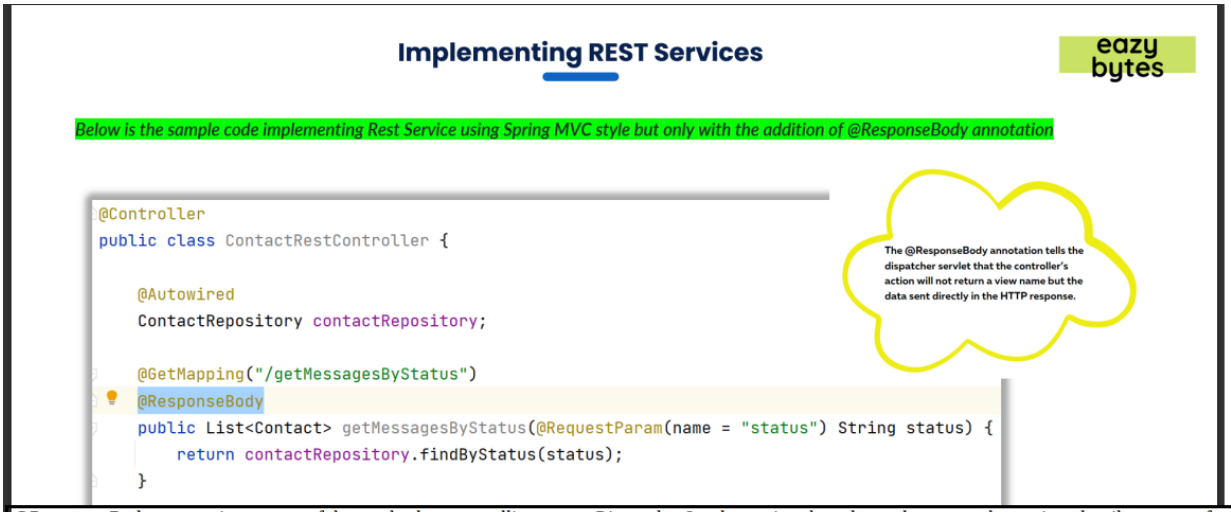


So the very first scenario is we have a backend server where we are exposing our backend logic or the data that we have inside a database with the help of REST web service and anymy mobile application I can make a request to this backend server using the REST service.
In this scenario, my mobile app is deployed into a separate server and my back end logic also deployed into a separate server.
So two different applications deployed in two different servers.
They are trying to communicate and exchange the data with the help of REST based services.

we have scenario two where two different backend applications which are hosted and deployed in two different servers.
They can communicate with each other by sending a request and receiving the response with the help of REST based web services.

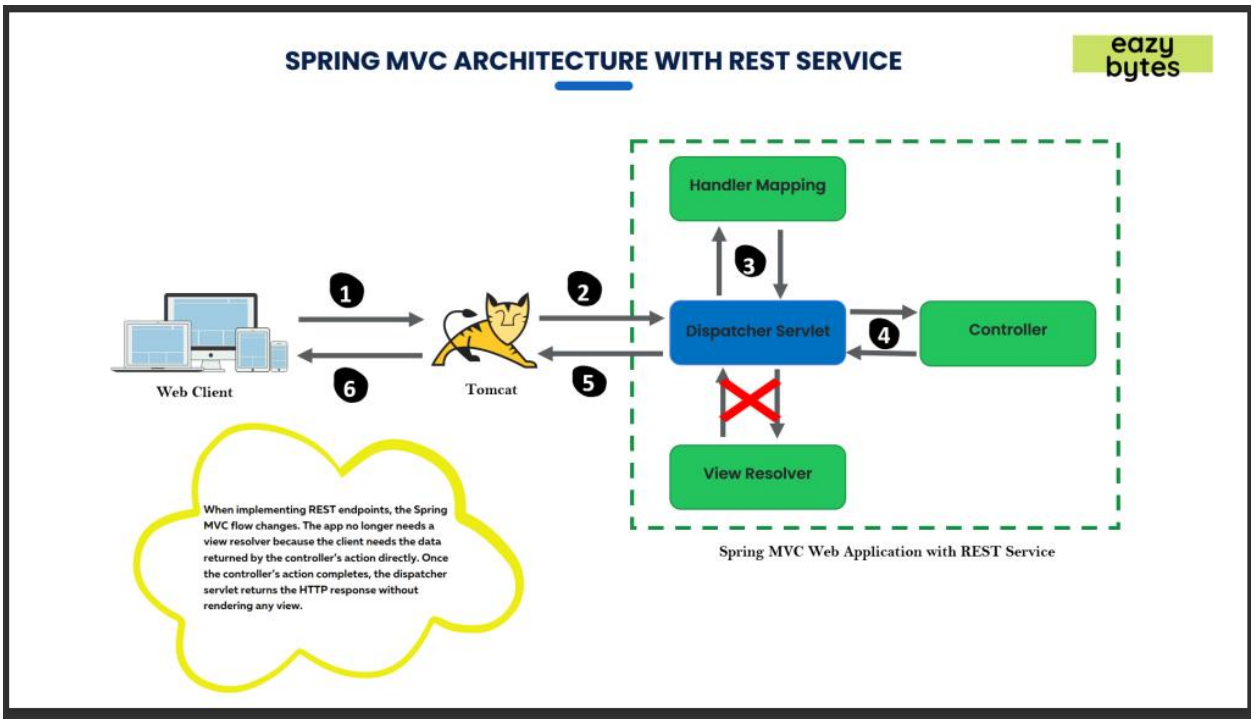
Third Scenario is there will UI team, they will keep developing their UI pages and all the navigation related changes, everything
using the frameworks like Angular and React.
But these UI frameworks has to interact with the database or with a backend server to execute a business logic or to get the data from a particular database.
So in these kind of full stack development scenarios, my backend team can build the REST web services based upon Java and Spring, and they can expose those REST based services to my UI application.
Now, my developer, they will make a rest call to my backend rest APIs with the help of Angular or React or any other UI framework that they are using.
So this way, though, my UI code is deployed in different server and my backend logic is deployed into different server, they can still communicate with each other
To the end user they will give an appearance such a way that everything is a single application and everything is deployed into a single server.

207: Building Rest Services using Spring MVC Style & @Response Body Annotation



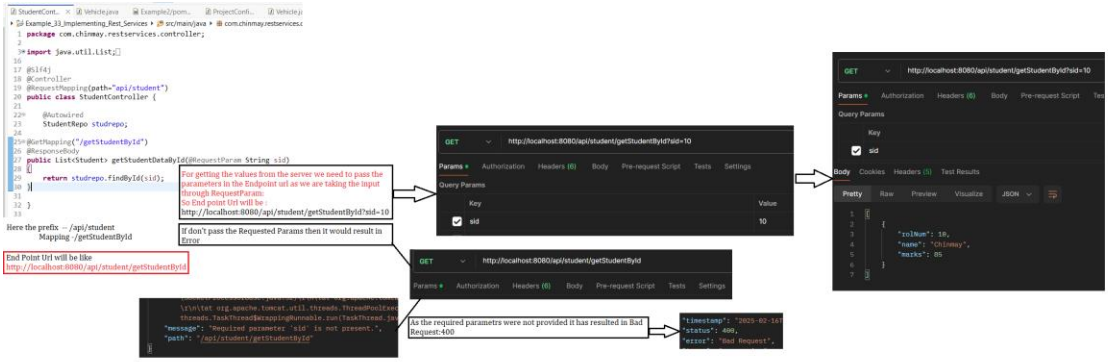
@ResponseBody annotation on top of the method we are telling to our DispatcherServlet saying that please do not send any view details as part of my response because
the other application which is trying to consume my REST service, they just need a data, they don't need a view.
They may have their own style of displaying the data.If it is a mobile application, they will take this data and display in their mobile application.
And similarly, if it is an another backend web application, they it will just accept the data or
If it some UI application is calling, which is based upon Angular or React, they will just take this data and they will take care of displaying the data inside a view.

So the response will be either in JSON /XML Format .



208: Implementing Rest Services using Spring MVC Style and @Response Body

Few Content of Spring Security is present refer that once if needed.



My spring framework or spring boot framework with the help Jackson Library, will convert this Java Pojo classes into an JSON format

210:Demo dive &Demo of @RequestBody**&Demo of @RequestBody Annotation**

```
@GetMapping(value="/getStudentByName")
@ResponseBody
public List<Student> getStudentDataByName(@RequestBody Student stu)
{
    if(stu!=null&&stu.getName()!=null)
    {
        return studrepo.findByName(stu.getName());
        //return studrepo.findAll();
    }
    else
    {
        return List.of();
    }
```

In our case what ever is the body that we are receving from the End user will be mapped with the Student POJO Object.
If the value is not provided it will be considered with default value
If the value is provided in the JSON body from the end user that value will be mapped to the corresponding varaible in the pojo Object.

whenever we want to accept the data that is coming inside the RequestBody,we need to use an annotation with the name @RequestBody

whenever we use @RequestBody, my spring framework will convert the JSON request that is coming from my end user(Ex:Postman) or from other application and it will try to assign that information to an Java POJO object mentioned(Student) in our case.

```
33 @GetMapping(value="/getStudentByName")
34 @ResponseBody
35 public List<Student> getStudentDataByName(@RequestBody Student stu)
36 {
37     if(stu!=null&&stu.getName()!=null)
38     {
39         return studrepo.findByName(stu.getName());
40         //return studrepo.findAll();
41     }
42     else
43     {
44         return List.of();
45     }
46
47 }
```

stu= Student (id=107)
- marks= 0
- name= "Chinmay" (id=121)
- rollNum= 0

```
Student [rollNum=0, name=Chinmay, marks=0]
```

GET http://localhost:8080/api/student/getStudentByName

Params Authorization Headers (8) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON

```
{
    "name":"Chinmay"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
[
    {
        "rollNum": 10,
        "name": "Chinmay",
        "marks": 85
    }
]
```

Response received when we have matching record in backend server.

GET http://localhost:8080/api/student/getStudentByName

Params Authorization Headers (8) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON

```
{
    "name":"Anil"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
[]
```

Empty Response received if no data is present for that name

211: Implement Rest Services using @RestController Annotation

Vehicle.java | Example2/pom... | ProjectConfi... | Vehicle.java | StudentCont...

Example_34_Implementing_Rest_Services_Rest_Controller ▸ src/main/java ▸ com.chinmay.restse

```java
package com.chinmay.restservices.controller;

import java.util.List;

@Slf4j
@RestController
@RequestMapping(path="api/student")
public class StudentController {

    @Autowired
    StudentRepo studrepo;

@GetMapping("/getStudentById")
//@ResponseBody
public List<Student> getStudentDataById(@RequestParam String sid)
{
    return studrepo.findById(sid);
}

@GetMapping(value="/getStudentByName")
//@ResponseBody
public List<Student> getStudentDataByName(@RequestBody Student stu)
{
    if(stu!=null&&stu.getName()!=null)
    {
        return studrepo.findByName(stu.getName());
        //return studrepo.findAll();
    }
    else
    {
        return List.of();
    }

}

}
```

Approach 1:
One Approaching for Building Rest Services using Spring MVC style.

@Controller on top of Class and @ResponseBody on top of Method.

Approach 2:
Building Rest Services is by specifiying @RestController on top of Controller.@RestController is Combination of @Controller+@ResponseBody:
Once @RestController is specified on top of Controller class there will be no need to mention on @ResponseBody on top of each Java Method.

@RestController -@Controller+@ResponseBody

Approach 2 is the recommened Approach

There will no difference in the way in which we will be sending the request to interact with the Rest Service

Implementing RestServices using @RestController Annotation.

212.Demo of Save Operation using Rest Service @Response Entity

@PostMapping Annotation is used when we want to Save Some information in Database.

Indicates that the Service will allow only HttpPost request.

All the data coming from the RequestBody will be stored in the POJO class

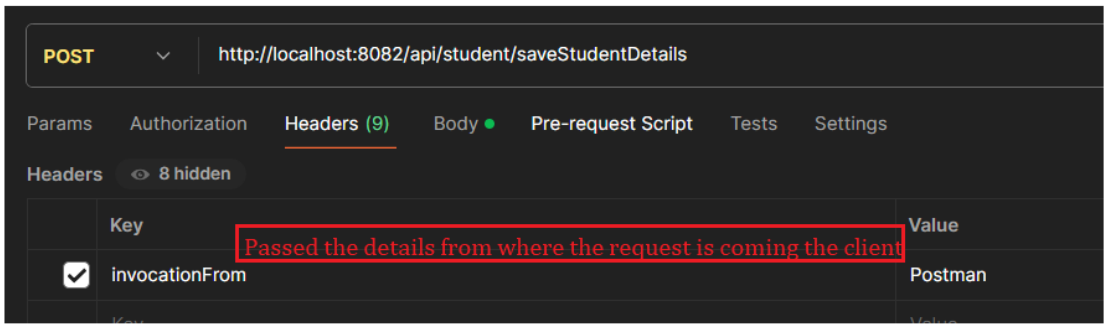
@RequestHeader we will be using when we want to know information like from whom we are receiving the request or from device we are receiving the client request.We can use @RequestHeader and get the details related to them.

public ResponseEntity<Response> saveStudentData(@RequestHeader("invocationFrom") String invocationFrom, @RequestBody Student stu)

```
@PostMapping(value="/saveStudentDetails")
public ResponseEntity<Response> saveStudentData(@RequestHeader("invocationFrom") String invocationFrom,
{
    log.info("Api is Invoked from "+invocationFrom);
```

```
82-exec-2] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherSe
82-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 2 ms
82-exec-2] c.c.r.controller.StudentController       : Api is Invoked from Postman
82-exec-2] com.zaxxer.hikari.HikariDataSource       : HikariPool-2 - Starting...
82-exec-2] com.zaxxer.hikari.pool.HikariPool        : HikariPool-2 - Added connection or
82-exec-2] com.zaxxer.hikari.HikariDataSource       : HikariPool-2 - Start completed.
```

postman Screenshot from where we are trying to access End point



@Valid Annotaion -when it is used along with the POJO class what ever are the validations present in the POJO Class they will also be validated.

Example_34_Implementing_Rest_Services_Rest_Controller

```
package com.chinmay.restservices.model;

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

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Response {
        private String statusCode;
        private String statusMsg;

}
```

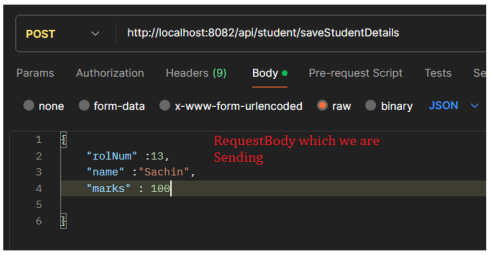
While sending the Response to the client if we want to send statusCode and header as well sent as response then we can use ResponeEntity class and append these to that class.

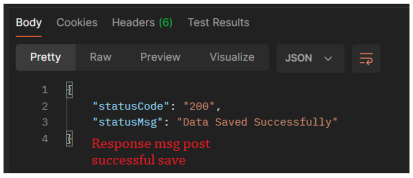
public ResponseEntity<Response>
Above we trying to retun an Response of Response model class so the same should be returned while returning at the End of Program.Rather if we try to return any other Model respone it would result in Error.

```
@PostMapping(value="/saveStudentDetails")
public ResponseEntity<Response> saveStudentData(@RequestHeader("invocationFrom") String invocationFrom, @RequestBody Student stu)
{
        studrepo.save(stu);
        Response resp=new Response();
        resp.setStatusCode("200");
        resp.setStatusMsg("Data Saved Successfully");
        return ResponseEntity.status(HttpStatus.CREATED).header("isMsgSaved", "true").body(resp);
}
```

sid [PK] integer	marks integer	sname text
1	90	Chinmay Sai
2	100	George
3	98	Kiran
4	98	Navin
5	95	Harsh
6	95	Nandha
7	85	Charan
10	85	Chinmay
11	85	Charan
12	98	Anil
13	100	Sachin

Student Table data





214: Demo of deletion Operation using Rest Service and Request Entity.

```java
@DeleteMapping(value="/deleteStudent")
public ResponseEntity<Response> deleteStudent(RequestEntity<Student> request)
{
        HttpHeaders headers=request.getHeaders();
        //headers.entrySet();
        Set<String> st=headers.keySet();
        Iterator<String> it=st.iterator();
        while(it.hasNext())
        {
                System.out.println("Key :"+it.next()+" Value :"+headers.get(it.next()));
        }

        Student student=request.getBody();
        studrepo.delete(student.getRolNum());
        Response response=new Response();
        response.setStatusCode("200");
        response.setStatusMsg("Deleted Successfully");
        return ResponseEntity.status(HttpStatus.OK).body(response);

}
```

For deletion Operation we need to use DeleteMapping

Similar to ResponseEntity we have RequestEntity using which we can take the input header and body using RequestEntity .

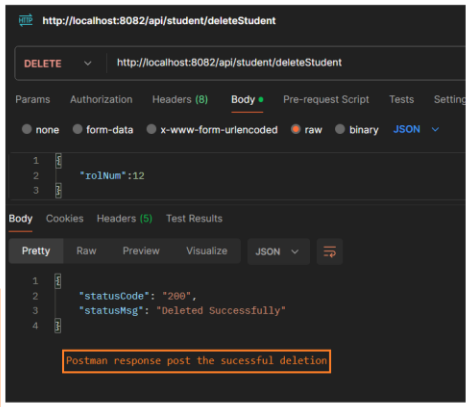
In the above we could see RequestEntity<Student> request is taking headers and body from the RequestEntity.

Headers:

```java
HttpHeaders headers=request.getHeaders();
        //headers.entrySet();
Set<String> st=headers.keySet();
```

Body :

```java
Studnet stu=request.getBody() //Here the value from the body will be mapped to the POJO Object
```



	sid [PK] integer	marks integer	sname text
1	1	90	Chinmay Sai
2	2	100	George
3	3	98	Kiran
4	4	98	Navin
5	5	95	Harsh
6	6	95	Nandha
7	7	85	Charan
8	10	85	Chinmay
9	11	85	Charan
10	13	100	Sachin

Database records post sucessful deletion

```java
public void delete(int sid)
{
    String sql="DELETE FROM STUDENT WHERE SID = ?";
    int j=jdbcTemplate.update(sql,sid);
    System.out.println("Deletion ");

}
```

StudentRep Delete code

214:Demo of Update Operation using Rest Service and recap of all Annotations.

```java
@PatchMapping(value="/updateStudent")
public ResponseEntity<Response> updateStudent(RequestEntity<Student> request)
{
        Response res=new Response();
        Student stud=request.getBody();
 List<Student> list=studrepo.findById(String.valueOf(stud.getRolNum()));
 if(list.size()==0)
 {
        res.setStatusCode("400");
        res.setStatusMsg("Invalid RollNum Received");
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(res);
 }
        studrepo.update(stud.getRolNum(),stud.getMarks());
        res.setStatusCode("200");
        res.setStatusMsg("Specified Row Updated Successfully");
        return ResponseEntity.status(HttpStatus.OK).body(res);
}
```

StudentController

```java
public void update(int sid,int marks)
{
    String sql="UPDATE STUDENT SET MARKS =? WHERE SID=?";
    int updatedrows=jdbcTemplate.update(sql,marks,sid);
    System.out.println("No of rows Affected"+updatedrows);
}
```

StudentRepo

PATCH http://localhost:8082/api/student/updateStudent

Params Authorization Headers (8) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON

```
{
  "rolNum":11,
  "marks":91
}
```

Sucess Scenario

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
  "statusCode": "200",
  "statusMsg": "Specified Row Updated Successfully"
}
```

Sucesfull Updation

PATCH http://localhost:8082/api/student/updateStudent

Params Authorization Headers (8) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON

```
{
  "rolNum":15,
  "marks":91
}
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
{
  "statusCode": "400",
  "statusMsg": "Invalid RollNum Received"
}
```

Bad Request as the Rollnumber is not part of table

214 : Implementing Global Error Logic for Rest Services using @RestControllerAdvice

Example_34_Implementing_Rest_Services_Rest_Controller ▸ src/main/java ▸ com.chinma

```java
package com.chinmay.restservices.controller;

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

import lombok.extern.slf4j.Slf4j;

@Slf4j
@RestControllerAdvice(annotations=RestController.class)
public class GlobalExceptionRestController {

}
```

So above screenshot means whatever global error logic that we define inside this class will be applicable for any exceptions that is going to happen inside the classes that are annotated with this annotation @RestController.

```java
public class GlobalExceptionRestController {

    @ExceptionHandler({Exception.class})
        public ResponseEntity<Response> exceptionHandler(Exception exception){
            Response response = new Response("500",
                    exception.getMessage());
            return new ResponseEntity(response, HttpStatus.INTERNAL_SERVER_ERROR);
        }
}
```

To implement the global error logic inside this class.we need to write a method with an annotation @ExceptionHandler and inside this method you can see I'm accepting the exception that happened as an method input parameter,and based upon that I'm populating the Response object with the values status code as 500 and the message whatever I have inside this exception by calling the exception.getMessage and post that I'm sendinga ResponseEntity as a return value with the response populated and at the same time I'm also sending the overall HTTP status with the help of internal server error.

```java
@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
                                                              HttpHeaders headers, HttpStatusCode statusCode, WebRequest request) {

    Response response = new Response(statusCode.toString(),
            ex.getBindingResult().toString());
    return new ResponseEntity(response, HttpStatus.BAD_REQUEST);
}
```

If want to define a logic that can get executed and send a response in the scenarios. Whenever someone is sending invalid data format to my rest APIs for the same, I need to extend a class with the name ResponseEntityExceptionhandler.
we extend ResponseEntityExceptionhandler class and override a method inside our GlobalExceptionRestController class.
The method name is handleMethodArgumentsNotValid().
So this method will get invoked in the scenarios. Whenever someone is trying to send an invalid input data to our rest services.
So in those kind of scenarios we can see I'm simply populating the Response object with the status that I have received and at the same time I'm trying to fetch the exception details by using getBindingResults.toString() that is available inside this MethodArgumentNotValidException.
And after that we just have to return the ResponseEntity by populating this response along with the HttpStatus as BAD_REQUEST.
So this way I have defined the global error logic for my rest API services.

```java
@Slf4j
@RestControllerAdvice(annotations=RestController.class)
@Order(1)
public class GlobalExceptionRestController {

    @ExceptionHandler({Exception.class})
        public ResponseEntity<Response> exceptionHandler(Exception excep
```

If a project has multiple exception handlers we can specify we can specify priority with the help of @Order Annotation.
@Order(1) -- This class will get higher priority

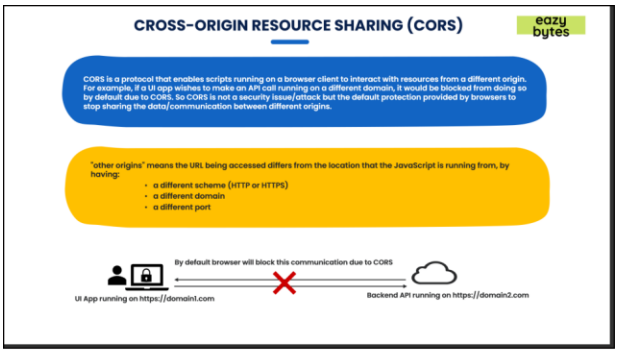
GlobalExceptionRestController	Refer this class for any queries in the above

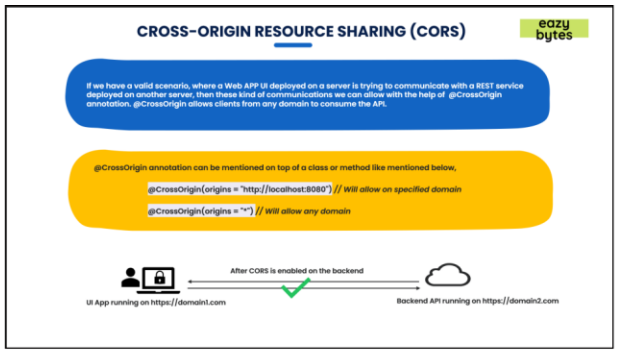
Different Annotation & Classes in REST

eazy bytes

- @RestController – can be used to put on top of a call. This will save developers from mentioning @ResponseBody on each methods
- @ResponseBody – can be used on top of a method to build a Rest API when we are using @Controller on top of a Java class
- ResponseEntity<T> - Allow developers to send response body, status, and headers on the HTTP response.
- @RestControllerAdvice – is used to mark the class as a REST controller advice. Along with @ExceptionHandler, this can be used to handle exceptions globally inside app.
- RequestEntity<T> - Allow developers to receive the request body, header in a HTTP request.
- @RequestHeader & @RequestBody – is used to receive the request body and header individually.

215: Cross Origin Resource Sharing:





with the help of @CrossOrigin annotation ,we can let our browsers know to allow the cross communication between the multiple servers by disabling the CORS protection that we have by default inside our browsers.