Day13: Spring Boot Restfull webservices, REST API, JSON JACKSON, POSTMAN

@PathVariable:

The **@PathVariable** annotation is used to extract the value from the URI. It is most suitable for the RESTful web service where the URL contains some value. Spring MVC allows us to use multiple @PathVariable annotations in the same method. A path variable is a critical part of creating rest resources.

```
@GetMapping(value = "/student/{roll}") public Student getStudent(@PathVariable("roll")
Integer roll) { Student student = new Student(); student.setRoll(roll); // using path
variable student.setName("Ram"); student.setMarks(850); return student; } }
```

Now we need to give the request to our service with the path variable:

Example:

```
http://localhost:8088/studentapp/student/10
```

Defining multiple PathVariable:

```
package com.masai; import org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.PathVariable; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RestController; @RestController
@RequestMapping("/studentapp") public class HelloWorldController { @GetMapping(value =
"/student/{r}/{n}/{m}") public Student getStudent(@PathVariable("r") Integer roll,
@PathVariable("n") String name, @PathVariable("m") Integer marks) { Student student =
new Student(); student.setRoll(roll); // using path variable student.setName(name); //
using path variable student.setMarks(marks); // using path variable return student; } }
```

Give the request as follows:

http://localhost:8088/studentapp/student/10/Ram/500

@RequestParam:

This annotation is used to fetch query parameter/request parameter from the request.

Example:

```
http://localhost:8088/studentapp/student?name=Ram If more than one parameter we need to
pass then they need to be concataneted with & symbol
```

Example:

```
@GetMapping("/getStudent") public Student getStudentDetails(@RequestParam("roll")
Integer roll,@RequestParam String name,@RequestParam Integer marks) { return new
Student(roll, name, marks); }
```

http://localhost:7000/studentApp/getStudent?roll=100&name=Ram&marks=500

Note:- By default @RequestParam is mandatory, to make it optional :

```
@GetMapping("/getStudent") public Student getStudentDetails(@RequestParam(required =
false) Integer roll,@RequestParam(required = false) String name,@RequestParam(required =
false) Integer marks) { return new Student(roll, name, marks); }
```

Implementing Post method:

To implement the Post method, we need to use **@RequestBody** annotation.

This Post method is used to send some data (object) to our web-service method.

The @RequestBody annotation maps body of the web request to the method parameter. The body of the request is passed through an HttpMessageConverter. It resolves the method argument depending on the content type of the request.

Example:

```
@PostMapping(value = "/saveStudent",consumes = "application/json") public String
saveStudentDetails(@RequestBody Student student) { //here we can communicate with the
Service layer or Data Access Layer classes to //persist the Student object in the
Database. return "Student stored ,"+student.getName(); }
```

Here By default consume type is JSON only, so **consumes** = "**application/json**" is optional.

Example:

```
@PostMapping(value = "/saveStudent") public String saveStudentDetails(@RequestBody
Student student) { //here we can communicate with the Service layer or Data Access Layer
classes to //persist the Student object in the Database. return "Student stored
,"+student.getName(); }
```

Note: From the browser, we can send only the GET request, to send the POST request along with data we need to use **REST client,** like Postman software.

Download the Postman from https://www.getpostman.com/downloads/

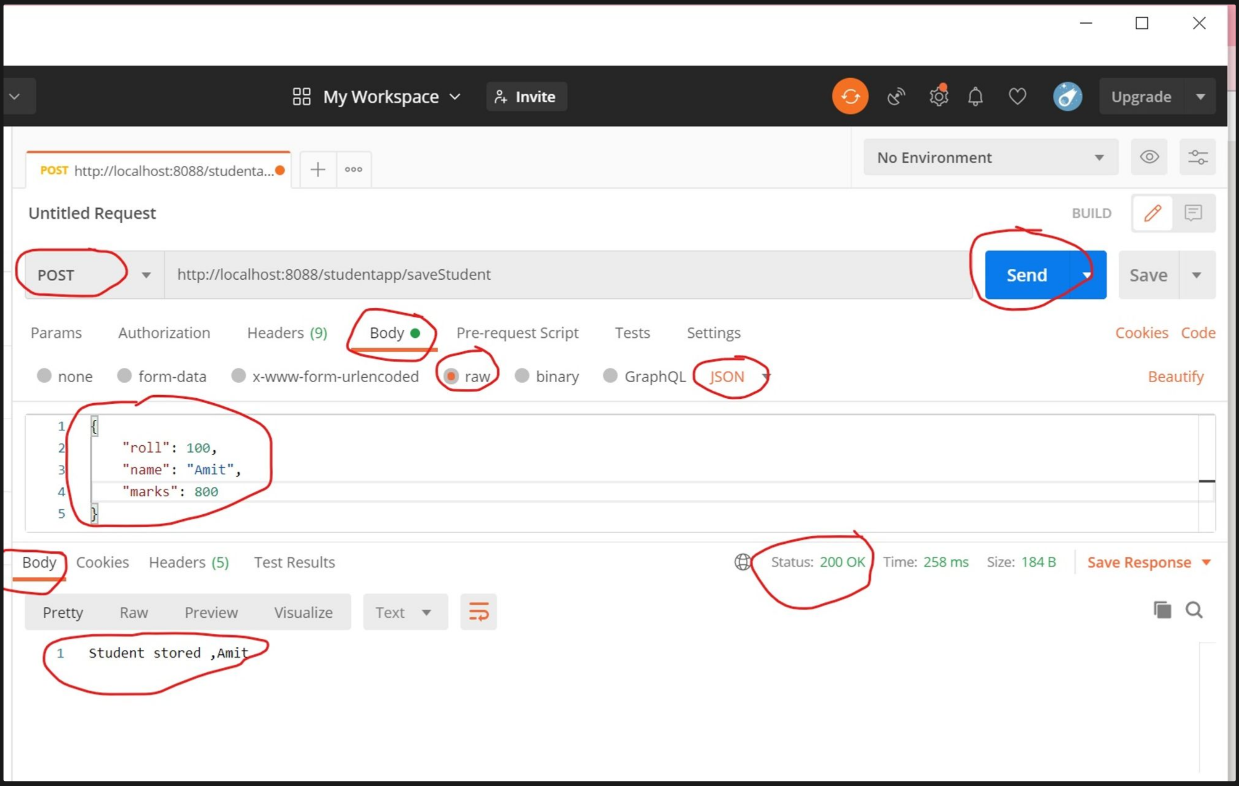
after installation, Launch the **Postman** and **Signup.**

From the Postman we need to send the POST request along with the Student object in the form of JSON object by using HTTP request Body.

Student Object in the form of JSON:

```
{ "roll": 50, "name": "Ratan", "marks": 750 }
```

Note: Here key should be in double quote.



POST http://localhost:8088/studentapp/saveStudent

```
{
    "roll": 100,
    "name": "Amit",
    "marks": 800
}
```

Status: 200 OK Time: 258 ms Size: 184 B

```
Student stored ,Amit
```

Implementing PUT method:

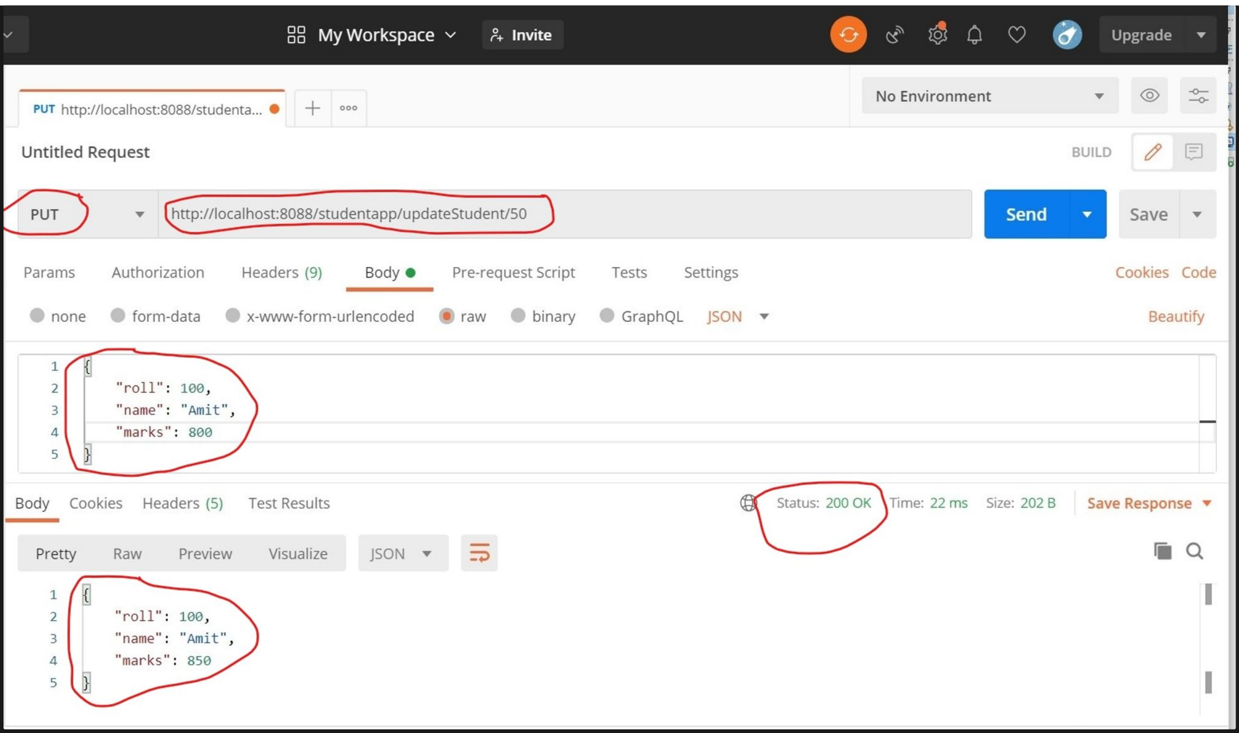
HTTP PUT method we use to update an existing resource:

Example:

```
@PutMapping(value = "/updateStudent/{graceMarks}") public Student
updateStudentDetails(@RequestBody Student student,@PathVariable("graceMarks")int gMarks)
{ student.setMarks(student.getMarks()+gMarks); return student; }
```

.

.



PUT http://localhost:8088/studentapp/updateStudent/50

```
{
    "roll": 100,
    "name": "Amit",
    "marks": 800
}
```

Status: 200 OK Time: 22 ms Size: 202 B

```
{
    "roll": 100,
    "name": "Amit",
    "marks": 850
}
```

ResponseEntity:

ResponseEntity **represents the whole HTTP response: status code, headers, and body**. As a result, we can use it to fully configure the HTTP response.

If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.

ResponseEntity is a generic type. Consequently, we can use any type as the response body:

Example:

```
@GetMapping("/hello") public ResponseEntity<String> hello() { return new
ResponseEntity<>("Hello World!", HttpStatus.OK); }
```

Since we specify the response status programmatically, we can return with different status codes for different scenarios:

Example:

```
@GetMapping("/getAge/{age}") public ResponseEntity<String> getAgeHandler(
@PathVariable("age") int yearOfBirth) { if (isInFuture(yearOfBirth)) { return new
ResponseEntity<>( "Year of birth cannot be in the future", HttpStatus.BAD_REQUEST); }
return new ResponseEntity<>( "Your age is " + calculateAge(yearOfBirth), HttpStatus.OK);
}
```

Additionally, we can set HTTP headers: (Http Headers are some extra information about the response to the client).

```
@PutMapping(value = "/updateStudent/{graceMarks}") public ResponseEntity<Student>
updateStudentDetails(@RequestBody Student student,@PathVariable ("graceMarks")int
gMarks) { student.setMarks(student.getMarks()+gMarks); HttpHeaders hh=new HttpHeaders();
hh.add("jwt", "sdnjnwjk3kj412321231sdf"); hh.add("user", "admin"); hh.add("hello",
"abc"); ResponseEntity<Student> re=new ResponseEntity<>(student,hh,HttpStatus.ACCEPTED);
return re; }
```