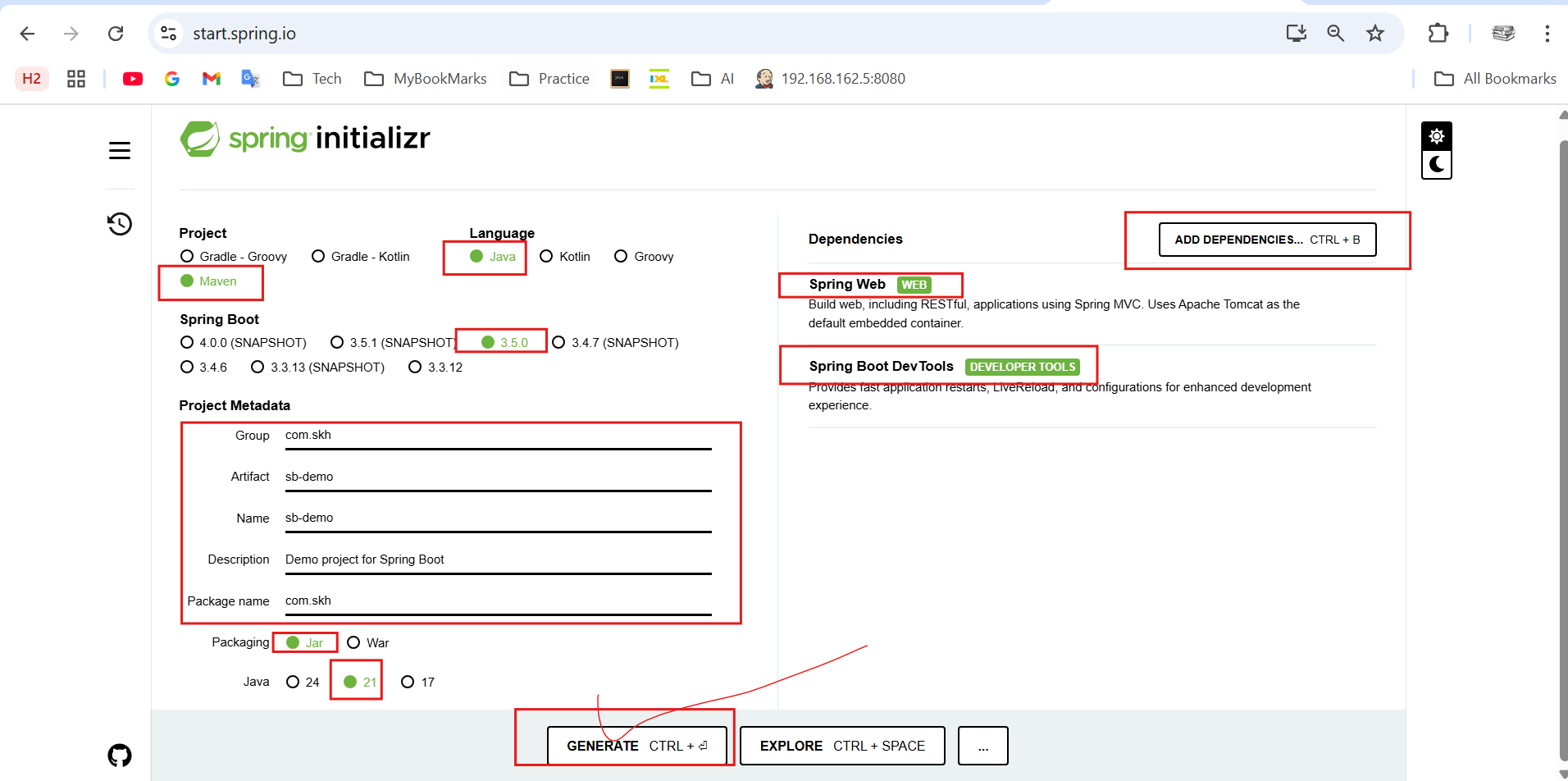
How to create Spring boot project?

Go to browser 🡪 open website called 🡪 <https://start.spring.io/> 🡪



If you don’t know java version installed in your computer.

Open command prompt and run > **java –version** it will display java version in your system.

Finally click on “**Generate**” button.

It will download the .zip file into downloads folder.

Go to Downloads folder and **extract** the .**zip** file. You will get the project folder.

From above:

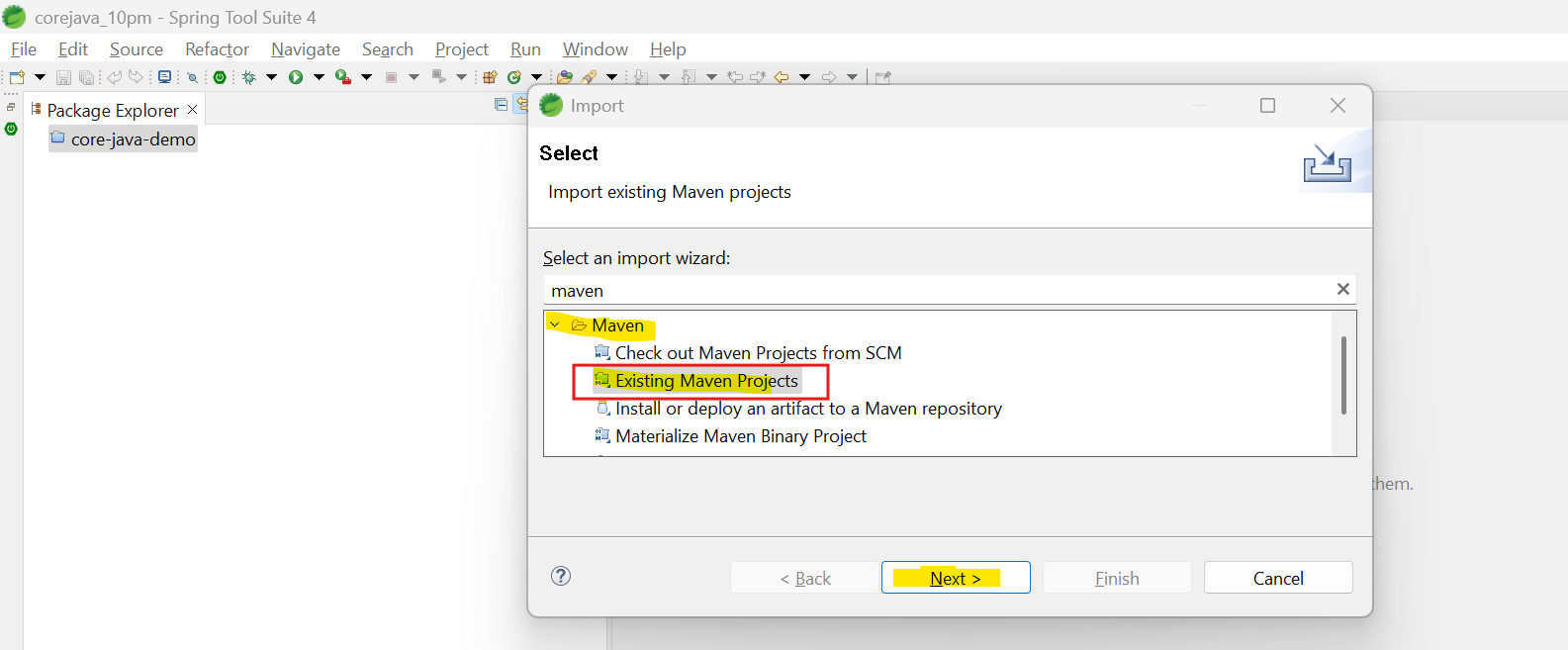
**Spring web** is for creating web application.

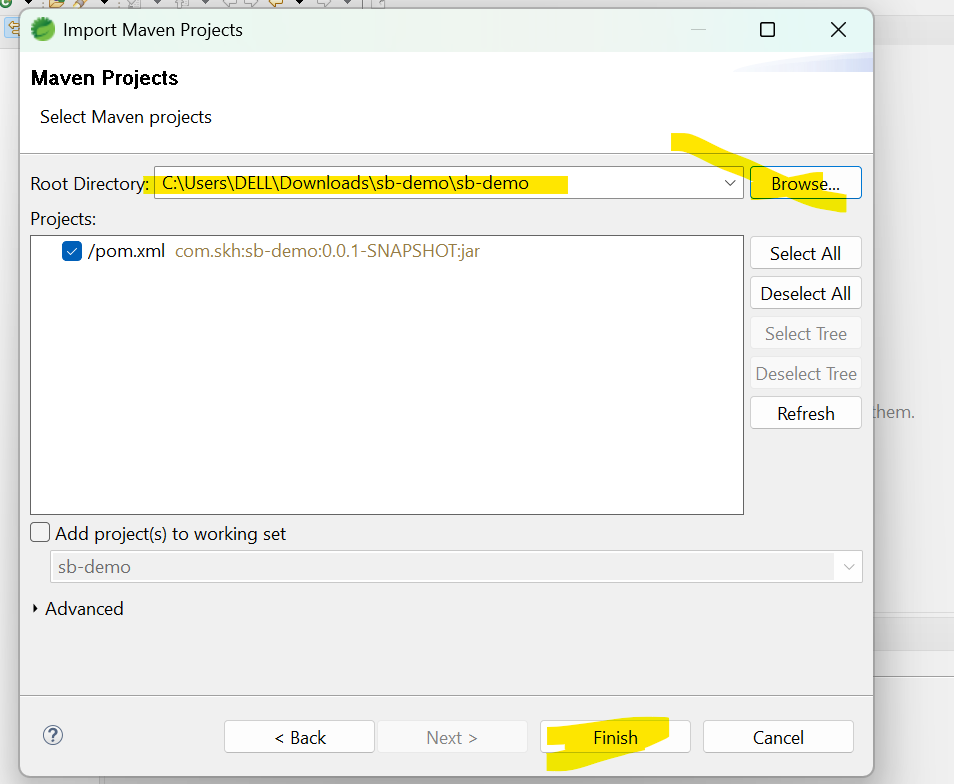
**Spring Boot** Dev Tools is for restarting the server automatically when any changes happened in code.

Now open STS / IntelliJ and import this new project there.

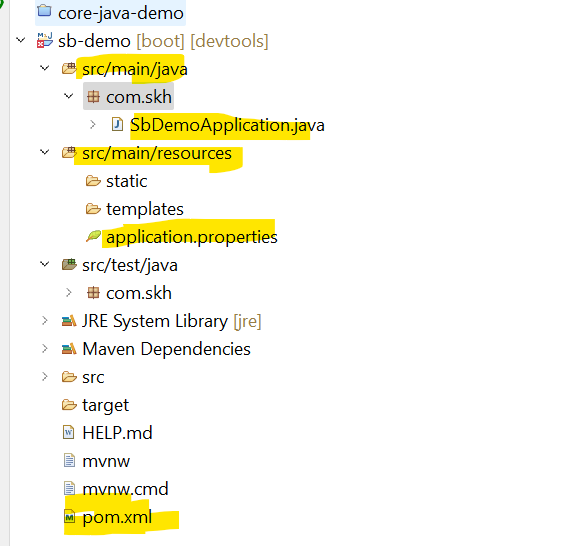
How to import new project into STS?

Open STS 🡪 File 🡪 Import 🡪 maven 🡪 Existing maven project 🡪 Next 🡪 Browser your project from downloads folder 🡪 click on Finish button.





After clicking project wait for few minutes. It will be downloaded from server.



**Project 1:**

Changes required to do:

1.**create a class** and add **@RestController** annotation on class. Eg: DemoController.java.

2.change name of **application.properties** file name into **application.yaml** file.[right click -> refractor-> rename]

3.[**OPTIONAL**]In the application.yaml file add server port number, number can be anything you interested. Like.. 9000,

9001, 9002…….etc.

4.create methods in a class you created, and add **Request method type annotations**….with **path** endpoint.

**@GetMapping(path=”/endpoint”), @PostMapping(path=”/endpoint”), @PutMapping, @DeleteMapping**..

5.start the server:

1st approach: right click on project 🡪 Run As 🡪 Spring boot App 🡪 server will start, you can see on console.

2nd approach: Open main class 🡪 right click on class 🡪 run as 🡪 java application.

6.Access the controller class method endpoint using **IP address** and **port number** and **method end point**.

<http://localhost:9000/fetchName>

here ->**http** is server **PROTOCOL**.

->**localhost** is **IP Address** of the server.

-> **9000** is **port number** of running server.

-> **fetchName** is controller method **endpoint**.

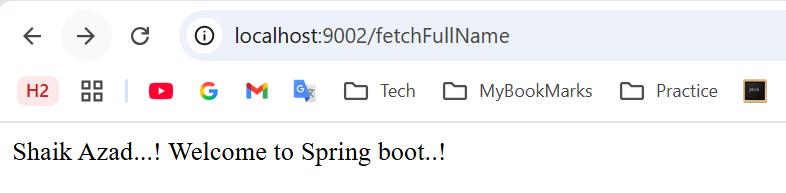
Go to Browser and use above URL.

**DemoController.java application.yaml**

|  |  |
| --- | --- |
| **package** com.skh;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.RestController;  **@RestController**  **public** **class** DemoController {    **@GetMapping(path = "/fetchName")**  **public** void m1() {  System.***out***.println("Hi this is Kamal..!");  }    @GetMapping(path = "/fetchNumber")  **public** **int** m2() {  **return** 123;  }    @GetMapping(path = "/fetchFullName")  **public** **String** m3() {  **return** "Shaik Azad...! Welcome to Spring boot..!";  }  } | server:  port: 9001 |

In application.yaml/yml 🡪 always maintain spaces properly.

If we don’t add port number in application.yaml file. Tomcat server by default takes 8080 as default port number.



**How many ways we can send the request to the server?**

There are so many ways are there.

1. Using browser URL
2. HTML
3. Javacript
4. **Postman – real time tools**
5. **jMeter – real time tools**

etc……

**How to send data from UI / browser to controller method?**

1. **Using path variable:** 🡪 URL**/inputValue 🡺**

[http://localhost:9002/fetchFullName/**John**](http://localhost:9002/fetchFullName/John)**....**

@GetMapping(path = "/fetchFullName**/{fName}")**

**@PathVariable** annotation in method parameter level.

|  |
| --- |
| @GetMapping(path = "/fetchFullName/{fName}")  **public** String m3(@PathVariable String fName) {  **return** String.*format*("Hi %s welcome to Spring boot", fName);  } |

1. **Using query String**. 🡪 URL**?key=value&key=value&key=valueY&key=value&….**

[**http://localhost:9002/sendQueryParams?name=kamal&city=hyd&pin=534456**](http://localhost:9002/sendQueryParams?name=kamal&city=hyd&pin=534456)

**@RequestParam**

Here : http – protocol

localhost – IP Address

9002 – PORT number.

sendQueryParams – Endpoint

?name=kamal&city=hyd&pin=534456 – Query String.

<http://localhost:9002/sendQueryParams?name=Arafth&city=mvg&pin=343434>

|  |
| --- |
| @GetMapping(path = "/sendQueryParams")  public String m4**(@RequestParam** String name, @RequestParam String city,  @RequestParam String pin) {  return String.*format*("My details %s - %s - %s ",name, city, pin);  } |

1. By submitting form.

|  |
| --- |
| @GetMapping(path = "/percentage**/{tmarks}/{hMarks}/{emarks}**") public Integer calculatePercentage(  **@PathVariable** Integer tmarks,  **@PathVariable** Integer hMarks,  **@PathVariable** Integer emarks) {  Integer percentage = ((tmarks + hMarks + emarks) \* 100) / 300;  return percentage; } |

We have seen problem with Path vriables. If we want o send more values we will send n path variables only but it will confuse us.

<http://localhost:9001/percentage/67/75/88/56/ssdfsdf/454543/sfsdfsdf/2242343>

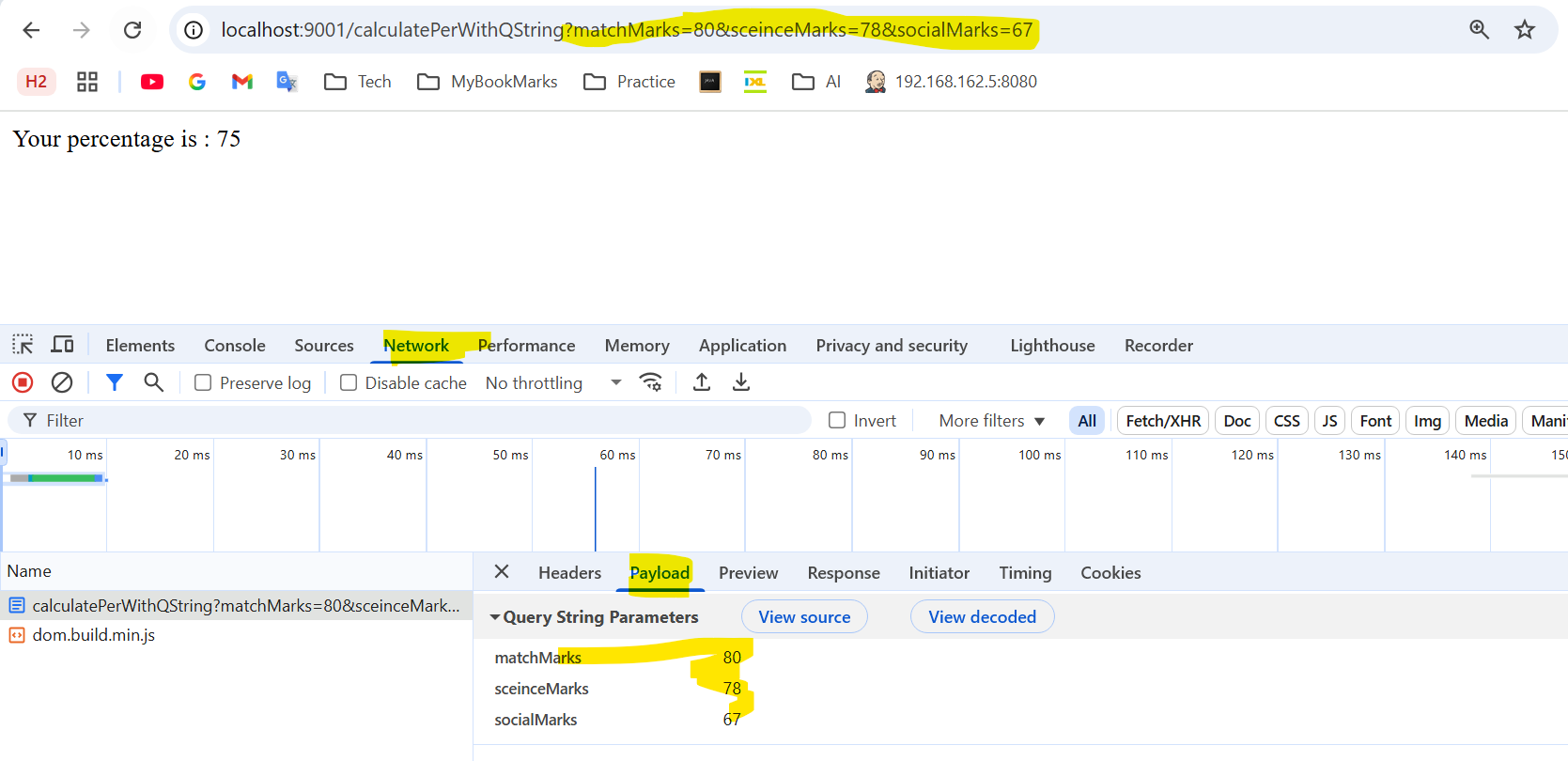
In this above URL, we don’t know which values is what for?

To overcome this we can use of “Query String” approach.

Payload means? 🡪 what ever the data we are sending to server is called as Payload.

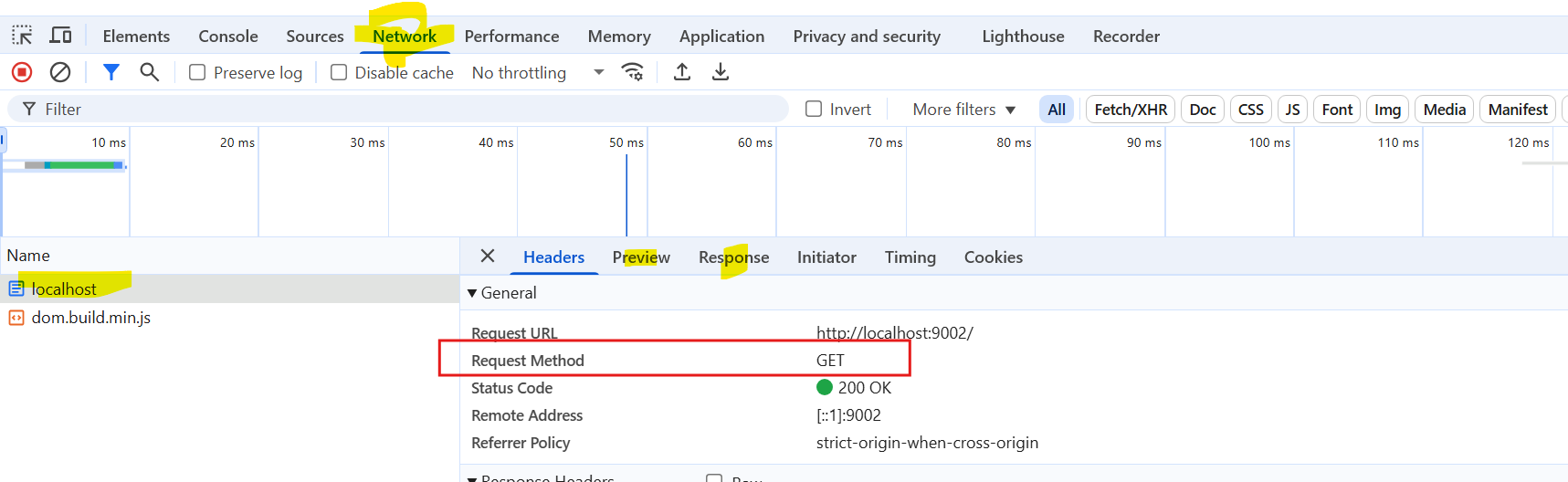
<http://localhost:9001/calculatePerWithQString?matchMarks=80&sceinceMarks=78&socialMarks=67>

|  |
| --- |
| @GetMapping(path= "/calculatePerWithQString") public String calculatePerWithQString(  **@RequestParam** Integer matchMarks,  **@RequestParam** Integer sceinceMarks,  **@RequestParam** Integer socialMarks) {  Integer percentage = ((matchMarks + sceinceMarks + socialMarks) \* 100) / 300;  return "Your percentage is : "+percentage; } |



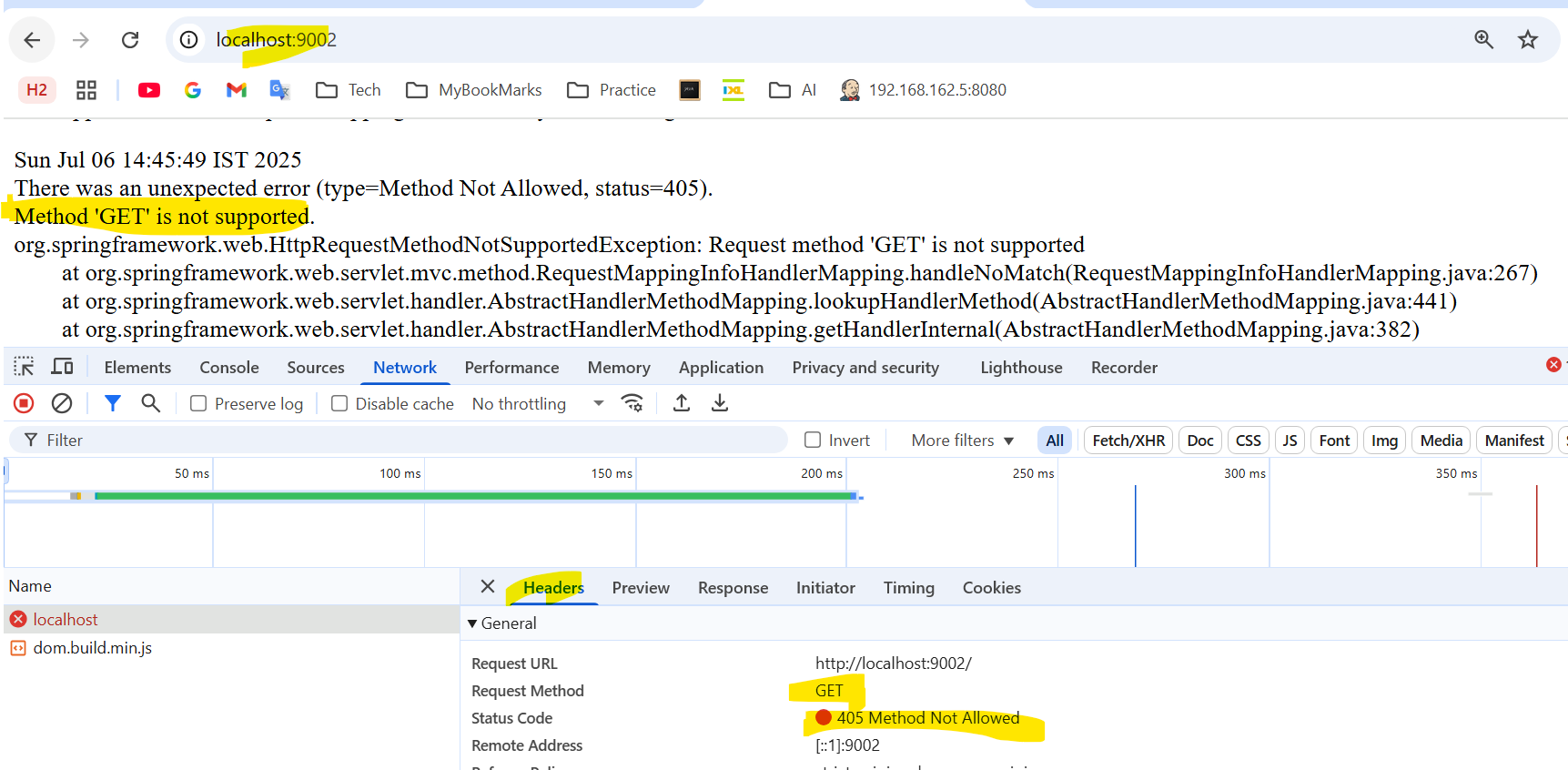
**When we send request from browser [normal request, path variable request, query string request], by default browser always sends “GET” request only.**

Particularly path variable and Query string approach always sends **GET** request only.



If we add @PostMapping in controller and if we send GET request from the browser, we will get below error on browser console.

|  |
| --- |
| @PostMapping("/")  **public** String defaultDisplay() {  **return** "Welcome to Spring boot...! This is Dahsboard / Homepage";  } |



**How to send the POST request in Web application [servlets / JSP / Spring / Spring boot / Angular/ ReactJS]?**

1. By using html **<form method=”POST”>** we can send POST request. [while using **Angular** and **ReactJS**]
2. By using **Java script code,** we can send POST request. [while using Java script application]
3. By using **POSTMAN**, we can send POST request. [while working with **Spring boot**]
4. By using **jMeter**, we can send POST request. [while working with **Spring boot**]
5. Etc……

By default, browser always send GET request only, but for sending **POST, PUT, DELETE** ..etc kind of requests we need to use above approaches.

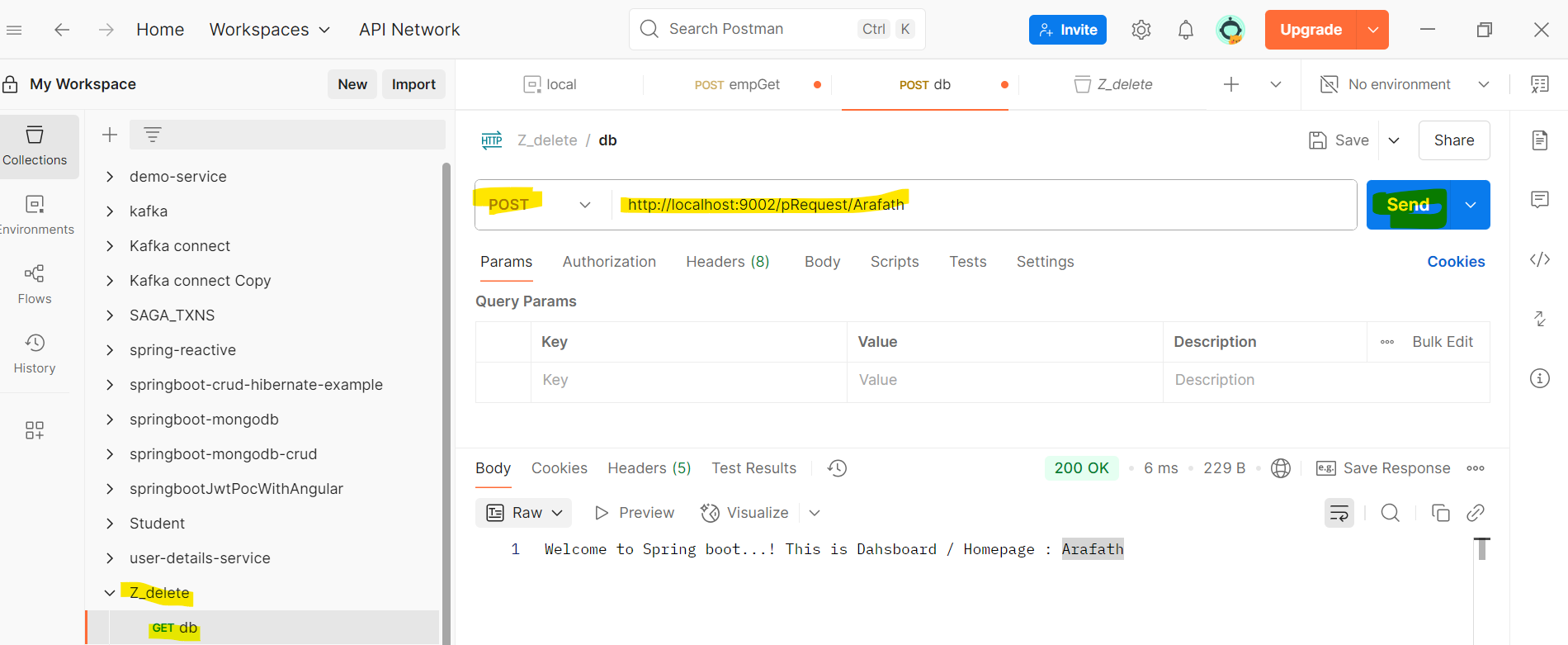
**Can we send path variables and query string in POST request?**

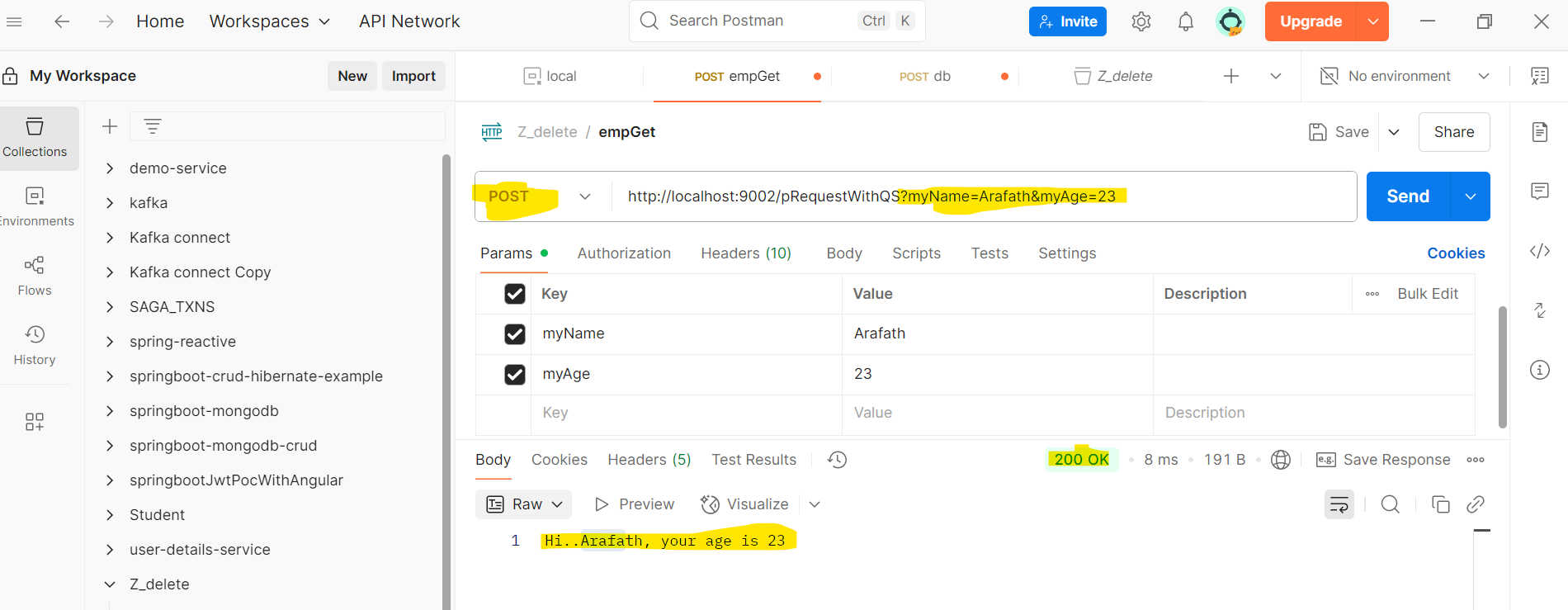
Yes, we can send. How we sent for GET request similar way we will send in POST request also.

The only difference is we use **@PostMapping** annotation in controller method and we use **POSTMAN** or **jMeter** for sending POST request.

Controller code:

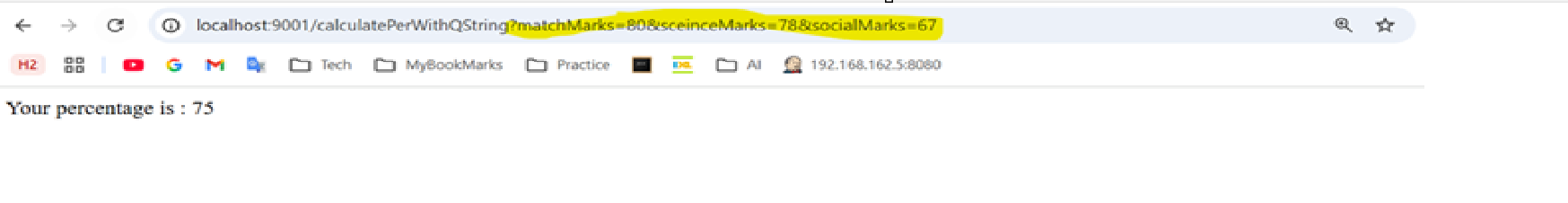
|  |
| --- |
| **@PostMapping("/pRequest/{myName}")**  **public** String defaultDisplay**(@PathVariable** String myName) {  **return** "Welcome to Spring boot...! This is Dahsboard / Homepage : " + myName;  }    **@PostMapping("/pRequestWithQS"**)  **public** String pRequestWithQS**(@RequestParam** String myName, **@RequestParam** Integer myAge) {  **return** String.*format*("Hi..%s, your age is %d", myName, myAge);  } |





But, while working with POST method it is not recommended to send path variables or query string.

We have drawbacks with path variables and query string, when we send data using path variables or query string that data is visible in browser URL bar.



We should not use path variables or query string for sensitive [passwords, bank account numbers, PAN, Aadhar, OTP…] data.

To overcome the drawbacks of GET request, they introduced POST request type.

To send data to server in secured manner by using POST method, we need to use payload / request body approach.

When we are working with path variable we used @PathVariable annotation and we use @RequestParam annotation for query string approach.

Now for sending request payload in **POST** request, in the controller we use **@RequestBody** annotation.

Path variables 🡪 @PathVariable

Query String 🡪 @RequestParam

Request payload in POST 🡪 @RequestBody

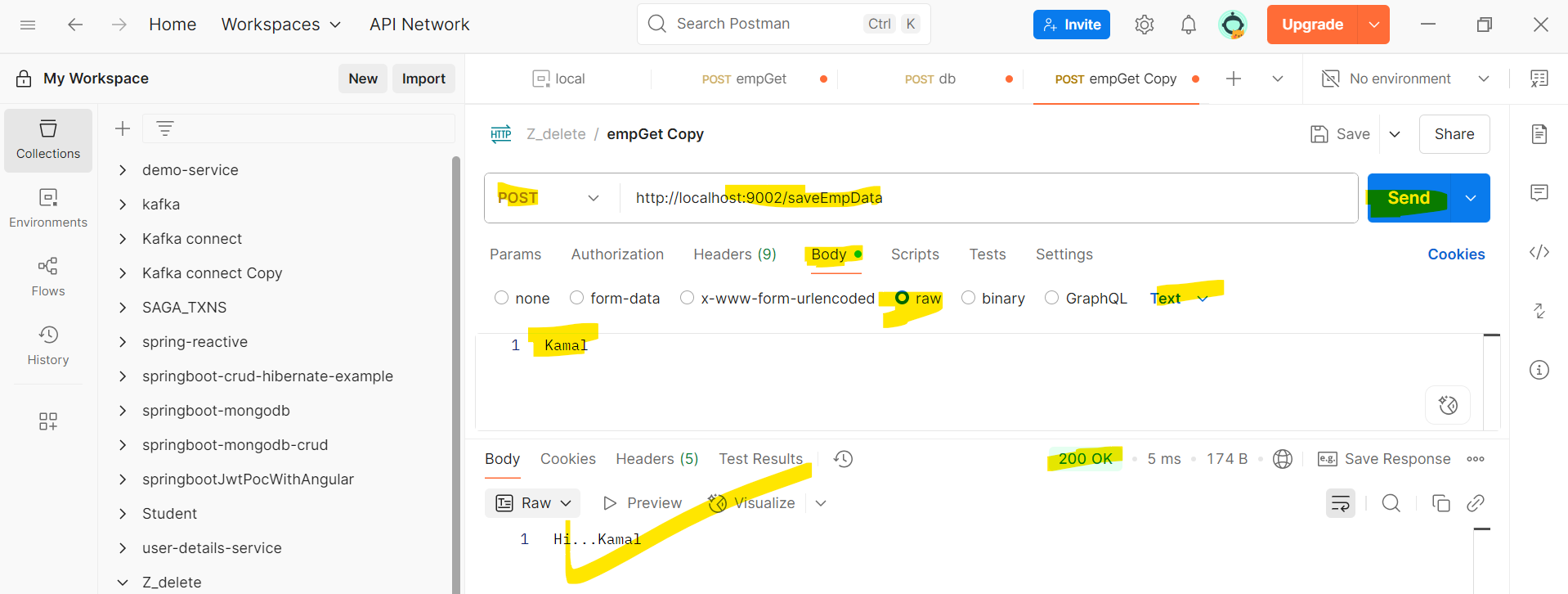
**How to send request payload from POSTMAN?**

|  |
| --- |
| **@PostMapping**(path = "/saveEmpData")  **public** String saveEmpData**(@RequestBody** String empName) {  **return** String.*format*("Hi...%s", empName);  } |

From POSTMAN. Send POST request.

Steps:

1. Select **Body** tab.
2. Select **raw** radio button.
3. Select TEST from below.
4. Pass value like shown below.



We can see above, what ever data we sent, it is not shown in URL bar, means we are sending data securely.

**How to send multiple data in POST request?**

Ans: We have to use POJO class.

**What is POJO (P**lain **O**ld **J**ava **O**bject**) class? How to create it? What is use of it?**

It is a plain java class which have only private fields, default constructor, setter methods, getter methods and toString() method.

For storing data in POJO class instance variables we use setter method.

For getting / fetching data from POJO class instance variables we use getter methods.

Pojo class will be used for storing the data transferring the data from one place to another place.

While sending data as POJO class, we need to send as JSON object from POSTMAN.

JSON object means data always send in {} braces, in key:value format. Keys will be double quotes, values will be according to data type.

|  |
| --- |
| {      "empName" : "SAAASS",      "empId" : 123  } |

For example:

|  |
| --- |
| **public** **class** EmployeePojo {  /\*  \* private fields default contructor setters & getters toString() method for  \* testing.  \*/  **private** String empName;  **private** Integer empId;  /\*  \* create setters and getters..  \* right click on file --> source --> click on "Generate getters and setters.."  \*  \*/    @Override  **public** String toString() {  **return** "EmployeePojo [empName=" + empName + ", empId=" + empId + "]";  }  } |

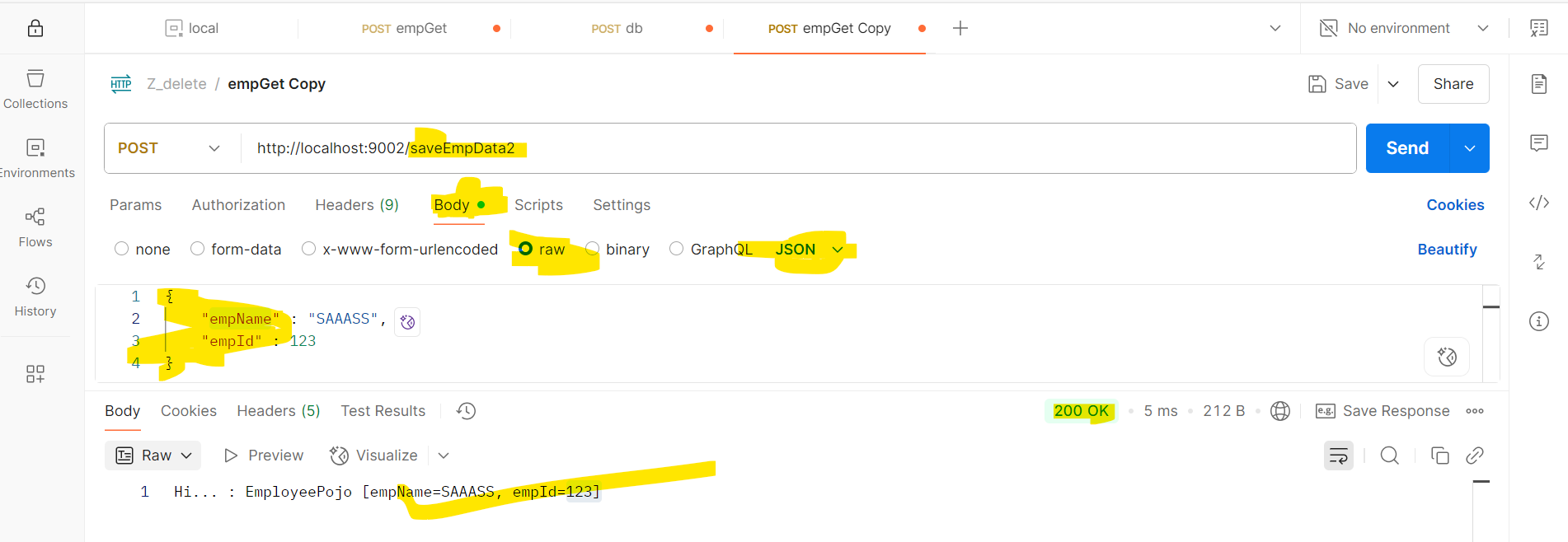
Binding?

Property name we sent and property name in pojo class matched then only biding happens and data will be given to pojo class property. If not matched it will take default values of pojo class property.

Controller code:

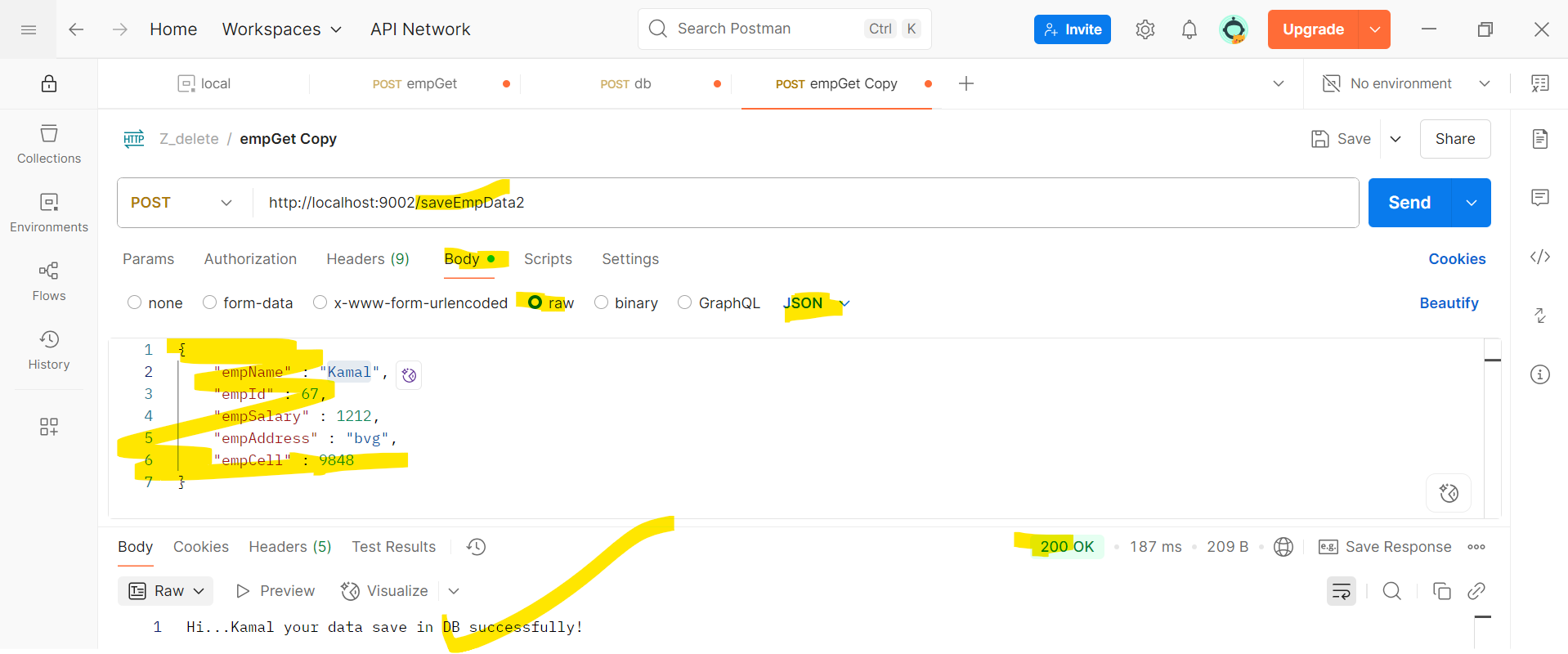
|  |
| --- |
| @PostMapping(path = "/saveEmpData2")  **public** String saveEmpData2(@RequestBody EmployeePojo empObj) {  **return** "Hi... : " + empObj;  } |

Now send request from POSTMAN.



Another example:

|  |
| --- |
| **@PostMapping**(path = "/saveEmpData2")  **public** String saveEmpData2**(@RequestBody** EmployeePojo empObj) {  System.*out*.println(empObj); // store in DB  return "Hi..." + empObj.getEmpName() + " your data save in DB successfully!" ;  } |



Based on your input data we need to create those many fields in POO class.

|  |
| --- |
| **public** **class** EmployeePojo {  **private** String empName;  **private** Integer empId;  **private** Double empSalary;  **private** String empAddress;  **private** Integer empCell;  // setters and getters  // toString() method.  } |

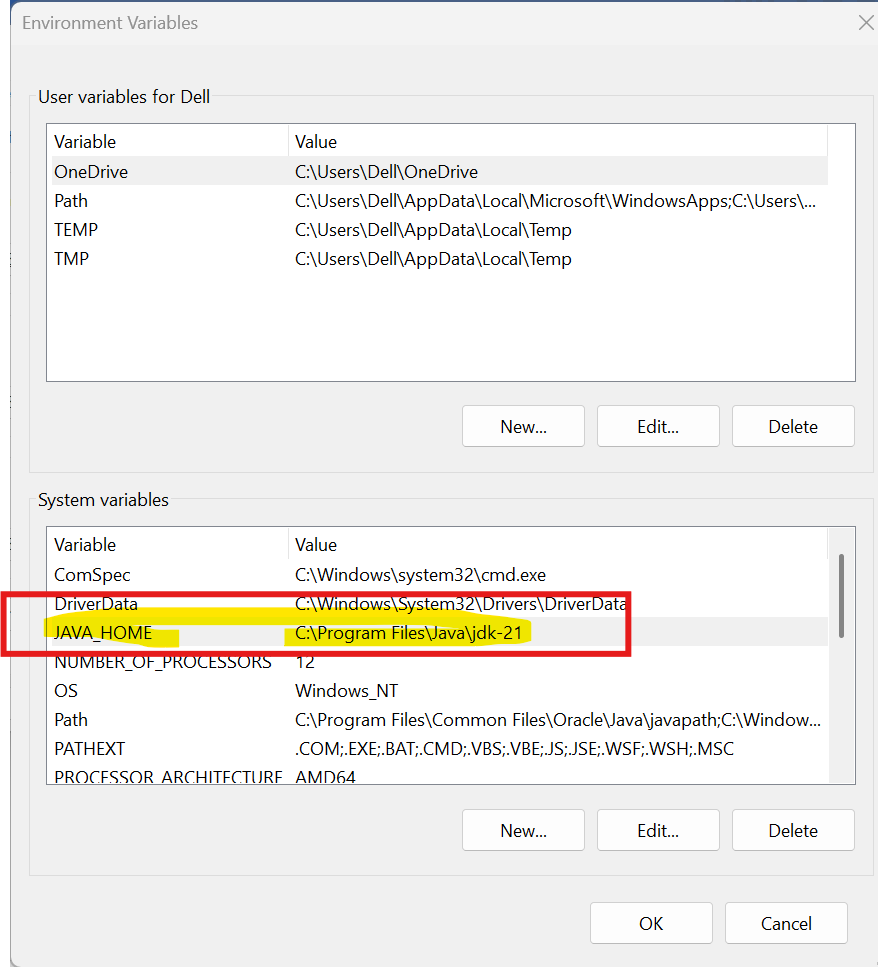
Complete controller example:

|  |
| --- |
| **@RestController**  **public** **class** DemoController {  @PostMapping("/pRequest/{myName}")  **public** String defaultDisplay(@PathVariable String myName) {  **return** "Welcome to Spring boot...! This is Dahsboard / Homepage : " + myName;  }  @PostMapping("/pRequestWithQS")  **public** String pRequestWithQS(@RequestParam String myName, @RequestParam Integer myAge) {  **return** String.*format*("Hi..%s, your age is %d", myName, myAge);  }  @PostMapping(path = "/saveEmpData")  **public** String saveEmpData(@RequestBody String empName) {  **return** String.*format*("Hi...%s", empName);  }  **@PostMapping(path = "/saveEmpData2")**  **public String saveEmpData2(@RequestBody EmployeePojo empObj) {**  **System.*out*.println(empObj); // store in DB**  **return "Hi..." + empObj.getEmpName() + " your data save in DB successfully!" ;**  **}**  } |

**MVC flow:**

**Tomcat Deployment:**

1. Download latest tomcat 🡪 <https://tomcat.apache.org/download-11.cgi>
2. Setting JAVA\_HOME is must to start tomcat server.



1. Add user related user name and password in “tomcat-user.xml” file in below path.

[D:\softwares\**apache-tomcat-11.0.10-windows-x64\apache-tomcat-11.0.10\conf**]

|  |
| --- |
| **tomcat-users.xml** |
| <user username="**admin**" password="**adminadmin**"  roles="**manager-gui,manager-script,manager-status,admin-gui**"/> |

**Sending GET request from JavaScript:**

|  |  |
| --- | --- |
| Test.html | UserCOntroller.java |
| <html>  <!DOCTYPE html>  <html lang="en">  <head>    <title>Employee Form</title>    <script src="test.js"></script>  </head>  <body>  **<button onclick="callMyFunction();">Click</button>**    <script>      function callMyFunction() {  **fetch('http://localhost:9000/api/users/dummyUserData')**  **.then(res => res.text())**  **.then(result => console.log(result))**  **.catch(error => console.error('Error:', error));**  **}**      let myFunction = (res) => res.text();      function f1(res) {        return res.text();      }      let f2 = (res) => {        return res.text();      }      let f3 = (res) => res.text();      let f4 = res => res.text();    </script>  </body>  </html> | package com.skh.controllers;  import com.skh.models.Employee; import com.skh.models.UserData; import jakarta.validation.Valid; import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.\*;  import java.util.List;  @RestController @RequestMapping("/api/users") **@CrossOrigin(origins = "\*")** public class **UserController** {     **@GetMapping(path = "/dummyUserData")**  public ResponseEntity<UserData> fetchDummyUserData() {  UserData userData = new UserData();  userData.setUserId("A123");  userData.setSessionId("B123");  return ResponseEntity.*ok*(userData);  }    } |

**Submitting form from JavaScript:**

**Index.html TestController.java**

|  |  |
| --- | --- |
| <!DOCTYPE html>  <html>  <body>      <form>          Employee Name: <input type="text" id="empName" value="Kamal" /> <br>          Employee Id: <input type="text"  value="12" /> <br>          Employee Sal: <input type="text"  value="1234" /> <br>          Employee Address: <input type="text"  value="BVG" /> <br>          Employee Cell: <input type="text" value="9848" /> <br>      </form>      <input type="submit" name="Submit" onclick="submitViaFetch()" /> <br>      <script>          function submitViaFetch() {              const employee = {                  empName: document.getElementById("empName").value,                  empId: 4554,                  empSalary: 4545,                  empAddress: "MVG",                  empCell: 454              }    **fetch**('http://localhost:9002/saveFullEmpDetailsFromJS', {                  method: 'POST',                  headers: {                      'Content-Type': 'application/json'                  },                  body: JSON.stringify(employee)              })                  .then(response => response.text())                  .then(result => console.log(result))                  .catch(error => console.error('Error:', error));          }      </script>  </body>  </html> | **@PostMapping**(path = "/saveFullEmpDetailsFromJS")  **public** String saveFullEmpDetailsFromJS(**@RequestBody** Map<String, String> empObj) {  System.***out***.println(empObj); // store in DB  **return** "Hi..." + empObj.get("empName") + " your data saved in DB successfully!" ;    } |