Spring Boot: its advance framework build on top of the Spring framework, with enhanced auto configurations, integrated servers

Spring modules: MVC, IOC, AOP, ORM or JPA, Security

IOC : Inversion of Control : by this module spring started controlling life cycle of object using xml files instead of using new operator with the help of dependency injection concepts.

Older days developer controlling the life cycle of objects, but in spring ioc objects life cycle control by spring ioc container.

Spring boot = spring - configuration + integrated server.

Older days we are doing manual configuration for beans:  
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="exampleBean" class="examples.ExampleBean">

<!-- setter injection using the nested <ref/> element -->

<property name="beanOne">

<ref bean="anotherExampleBean"/>

</property>

<!-- setter injection using the neater 'ref' attribute -->

<property name="beanTwo" ref="yetAnotherBean"/>

<property name="integerProperty" value="1"/>

</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>

<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>

</beans>

And also we are registering the packages with spring container for components scan purposs:

<**beans** ...> <**context:component-scan** **base-package="com.k7it"**/>

<**bean** class="com.baeldung.configuration.WebAppInitializer"/> </**beans**>

Spring boot = spring - manual configuration – manual component scan config + integrated server.

If we take any project it will have 3 layers

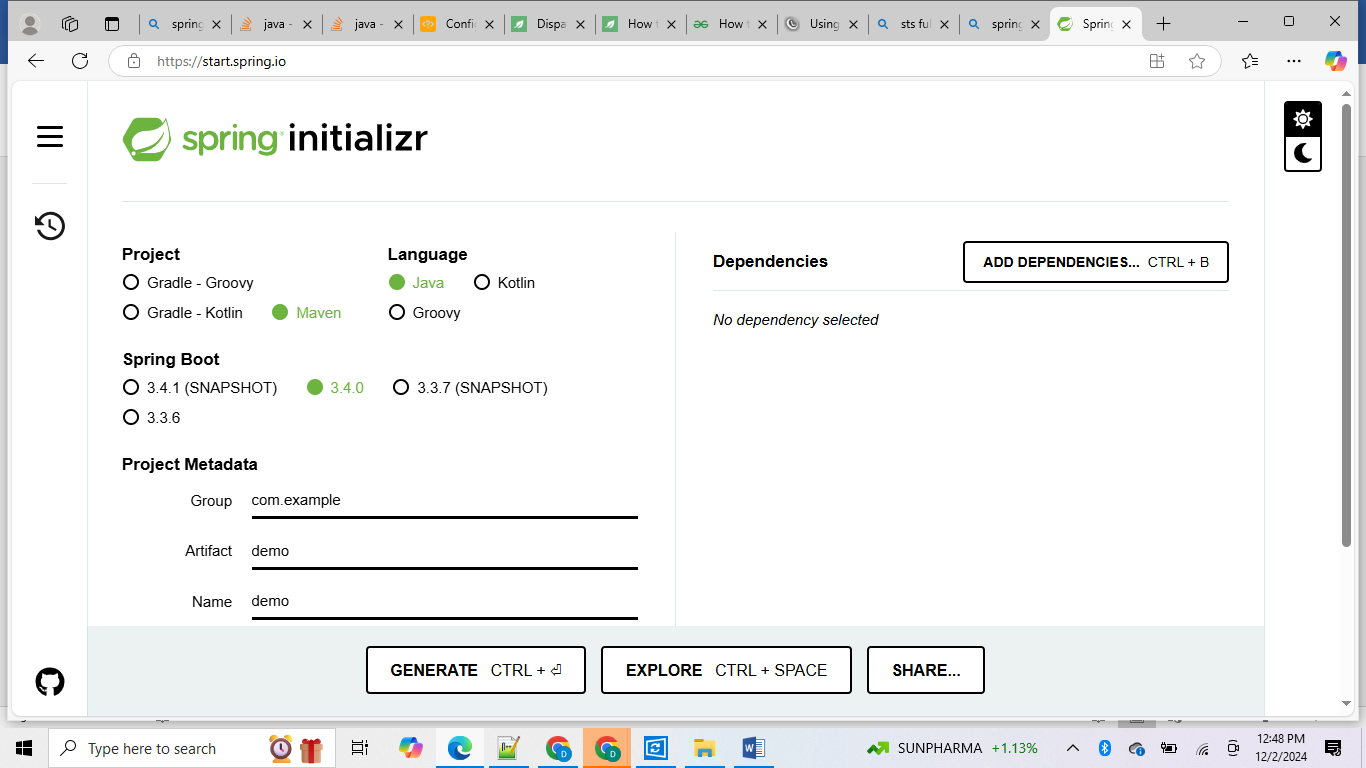
Presentation -> service layer -> database layer

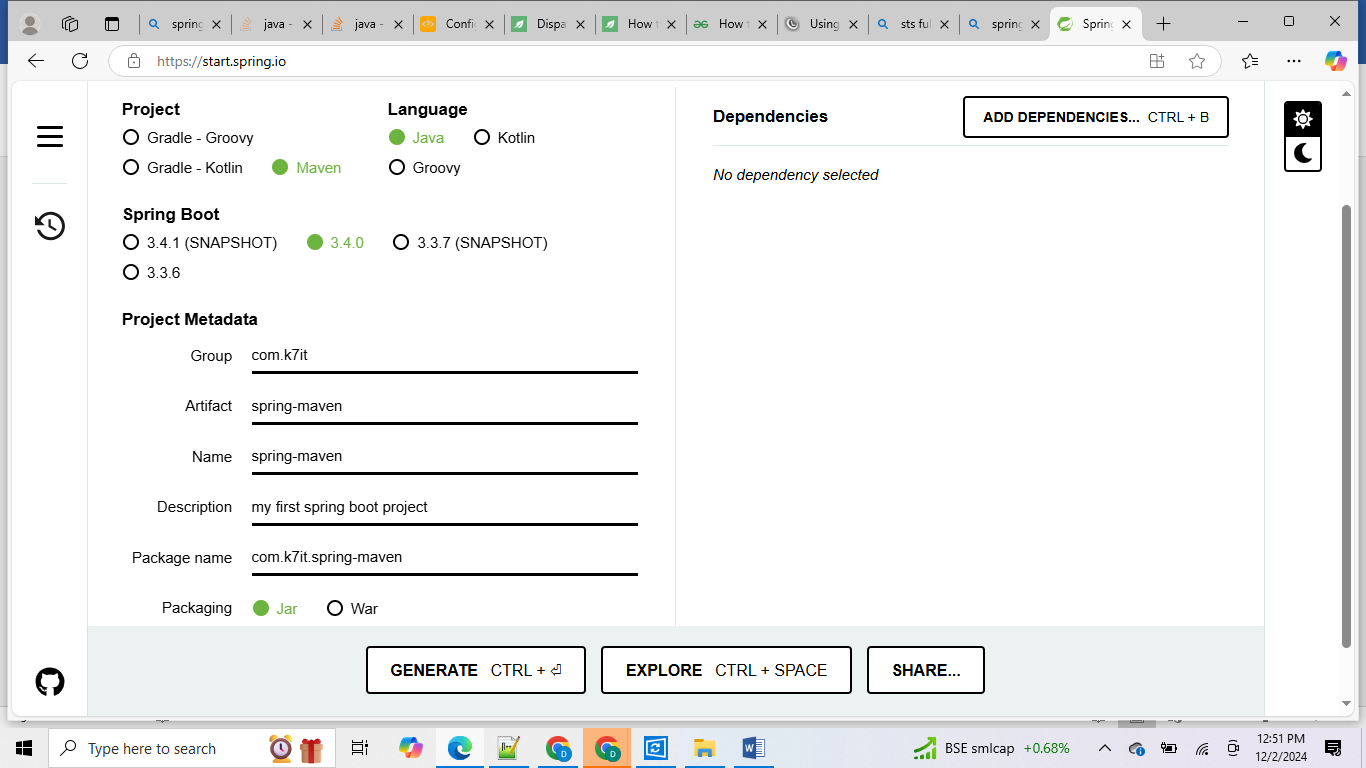
Html code -> server side logic or busniness logics -> database data

Html code –ULR/URI-> controller layer -> service layer(business logic) -> JPA(Repository) -> Database

Creating spring boot project: spring boot project is a one of maven or gradle projects, we can create spring boot projects 2 ways 1 using spring initializer web site or directly from our IDEs(eclipse or STS, Intelij). Here STS( Spring Tool Suite) is especially designed for spring project purpose .

1. First we will start with spring initializer web :  
   [Spring Initializr](https://start.spring.io/)





Next select the dependencies from right side: spring web and dev tools dependencies   
  
then click on generate button-> it will download zip file -> once it downa,od -> just perform unzip which folder u want.

Then open eclipse or STS -> file -> import project -> maven-> existing maven project ->

Browse your unzip folder location select root folder of pom.xml file -> click import.

After import complete we can see the main class in springboot project   
SpringMavenApplication.java

package com.k7it.spring\_maven;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringMavenApplication {

public static void main(String[] args) {

SpringApplication.run(SpringMavenApplication.class, args);

}

}

This is main class to start our spring boot project

Once u run this class it will build our spring boot project and deploy into default integrated server i.e tomcat server with default port is 8080.

Server logs:

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v3.4.0)

2024-12-02T13:03:49.403+05:30 INFO 9364 --- [spring-maven] [ restartedMain] c.k.spring\_maven.SpringMavenApplication : Starting SpringMavenApplication using Java 22.0.2 with PID 9364 (D:\lab\202402\_JavaBatch\spring\_dev\spring-maven\target\classes started by K7IT in D:\lab\202402\_JavaBatch\spring\_dev\spring-maven)

2024-12-02T13:03:49.408+05:30 INFO 9364 --- [spring-maven] [ restartedMain] c.k.spring\_maven.SpringMavenApplication : No active profile set, falling back to 1 default profile: "default"

2024-12-02T13:03:49.524+05:30 INFO 9364 --- [spring-maven] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable

2024-12-02T13:03:49.525+05:30 INFO 9364 --- [spring-maven] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'

2024-12-02T13:03:51.129+05:30 INFO 9364 --- [spring-maven] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)

2024-12-02T13:03:51.159+05:30 INFO 9364 --- [spring-maven] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2024-12-02T13:03:51.159+05:30 INFO 9364 --- [spring-maven] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.33]

2024-12-02T13:03:51.211+05:30 INFO 9364 --- [spring-maven] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2024-12-02T13:03:51.212+05:30 INFO 9364 --- [spring-maven] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1685 ms

2024-12-02T13:03:51.736+05:30 INFO 9364 --- [spring-maven] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729

2024-12-02T13:03:51.777+05:30 INFO 9364 --- [spring-maven] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : **Tomcat started on port 8080 (http) with context path '/'**

2024-12-02T13:03:51.790+05:30 INFO 9364 --- [spring-maven] [ restartedMain] c.k.spring\_maven.SpringMavenApplication : Started SpringMavenApplication in 3.182 seconds (process running for 3.827)

If we want to test this spring boot 1st we need to add our own controller class with one sample endpoint

Let create TestController class with @RestController annotation to make this class is spring boot controller class.

**package** com.k7it.spring\_maven.controller;

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

@RestController

**public** **class** TestController {

**public** String sayHelloWorld() {

**return** "Hello World this is my first spring end point";

}

}

Here if we want to interact with html to java there some method types based user operations

GET : getting info from back end to frontend  
POST : sending data from front end to back end  
PUT : updating exiting data in backend from front end   
DELETE: deleting existing back end data from front end  
PATCH : if we want to update only one or two fields partially in side one record we will go with patch method.

HEAD  
OPTION

### Spring Data JPA:

=================

* **Spring Data JPA Repositories:** Use repositories to interact with the database. These repositories provide methods like save(), findById(), findAll(), and deleteById() to perform CRUD operations.

In spring boot for every method type there is a one annotations

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

@PatchMapping

In side these annotations we need to provide what is the calling statement for our end point

Method from UI or front end or html , since we can’t access our method using method name or signature directly like java class to java class .

This calling statement name is called as URL or URI for that corresponding method.

Some times this URL or URI is depending one class URI or URL also . if class is having specific identification then we should use class identification and method identification to reach our method destination .

**package** com.k7it.spring\_maven.controller;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.RequestMapping;

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

@RestController

@RequestMapping("/api/v1")

**public** **class** TestController {

@GetMapping("/sayHello")// final end point class end point + method end point = "/api/v1/sayHello"

**public** String sayHelloWorld() {

**return** "Hello World this is my first spring end point";

}

}

Now this classes are running under server so we can’t access these classes directly without access server. If we want to access server we need IP address and port number of corresponding server.

If any of the severs are running in local those servers we will get default port number is   
127.0.0.1 or we can say localhost : here local host act like DNS name

And default tomcat server port number is 8080

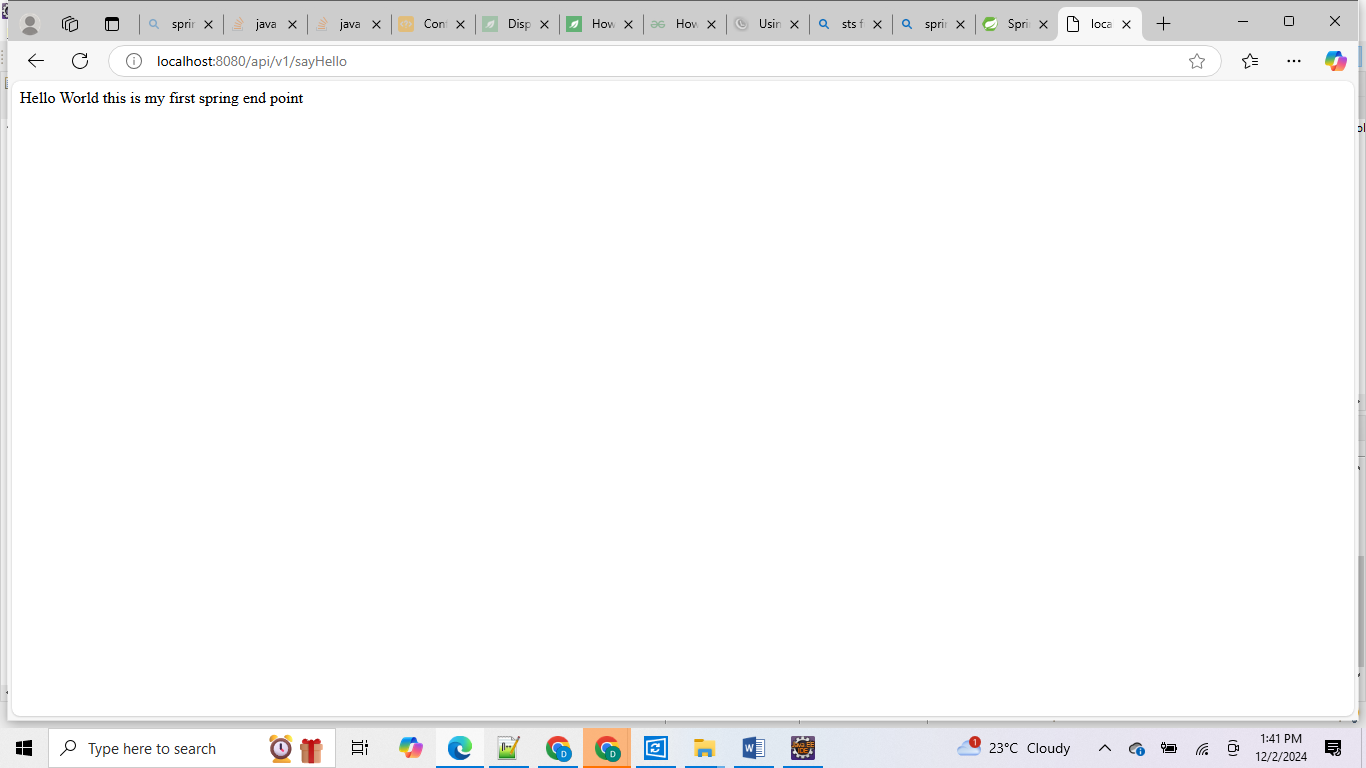
If we want access our server we use : <http://127.0.0.1:8080> or <http://localhost:8080>

If we want to access our end point method from this server we should use

Server url + out method URI or URL i.e

<http://127.0.0.1:8080/api/v1/sayHello> -> our sayHelloWorld method end point or API URL

if we type this url from browser we can will below output



Basically if our end points are GET method type we can test from browser directly but in case of POST or PUT or DELETE all type methods its very difficult to test direct from browser then we should take help from HTML code or POST MAN tool or swagger UI tool or other alternative tools for post man i.e

 **Insomnia REST Client**: A powerful and user-friendly REST API client with features like cookie management, environment variables, and code generation.

 **HTTPie**: A command-line HTTP client that is easy to use and supports JSON.

 **Hoppscotch**: A web-based API development tool that is lightweight and easy to use.

 **RapidAPI**: A platform that allows you to connect to thousands of APIs and manage them in one place.

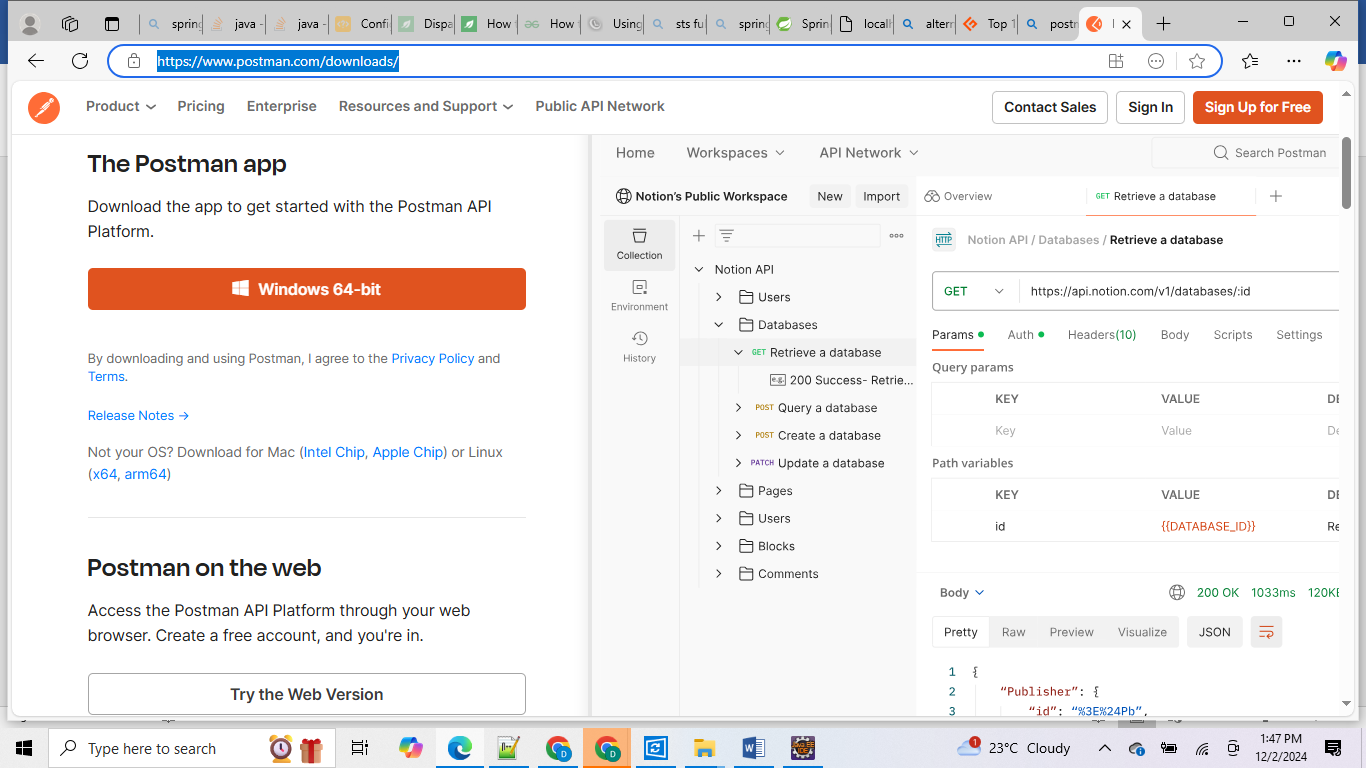
 **Testfully**: A web and desktop application for API testing that offers a clean interface and powerful features.

 **Thunder Client**: A lightweight REST API client extension for Visual Studio Code.

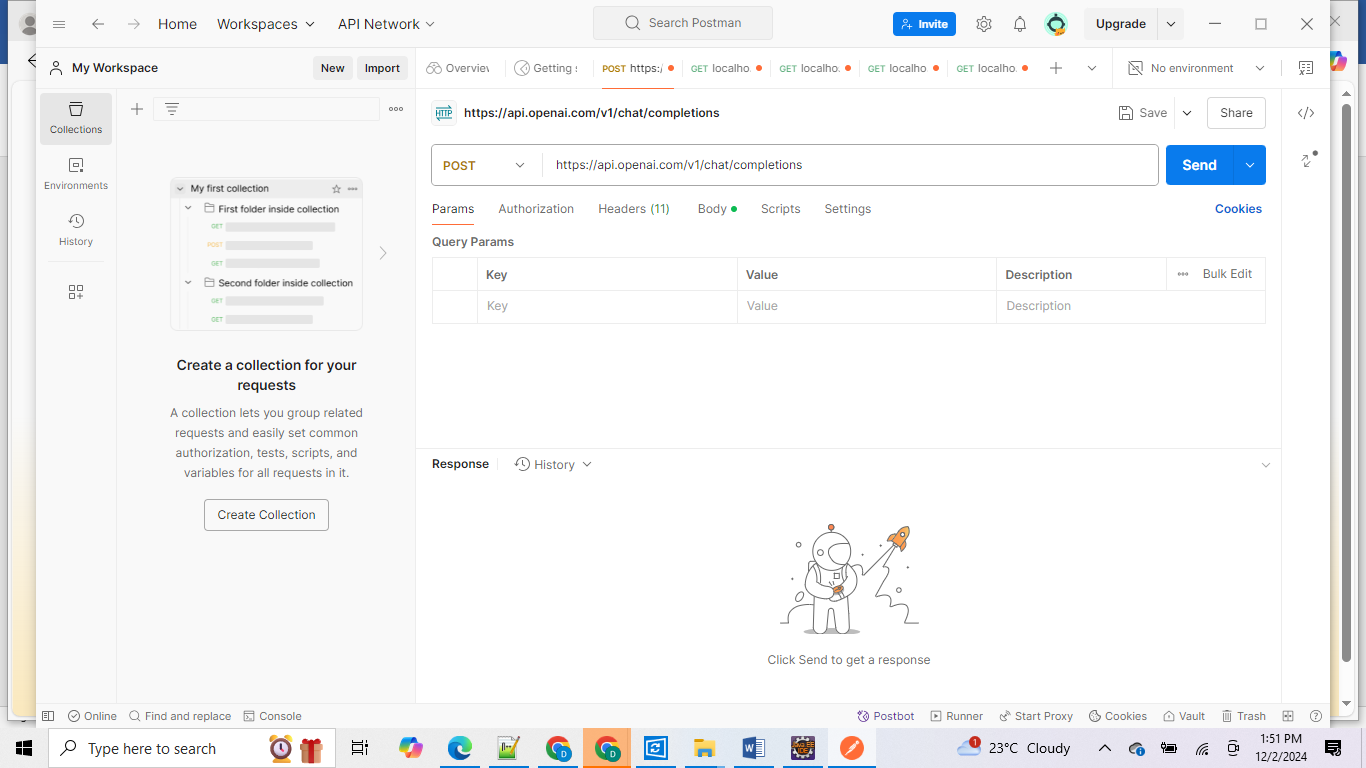
 **SoapUI :** A tool for testing SOAP and REST APIs with advanced features for functional testing

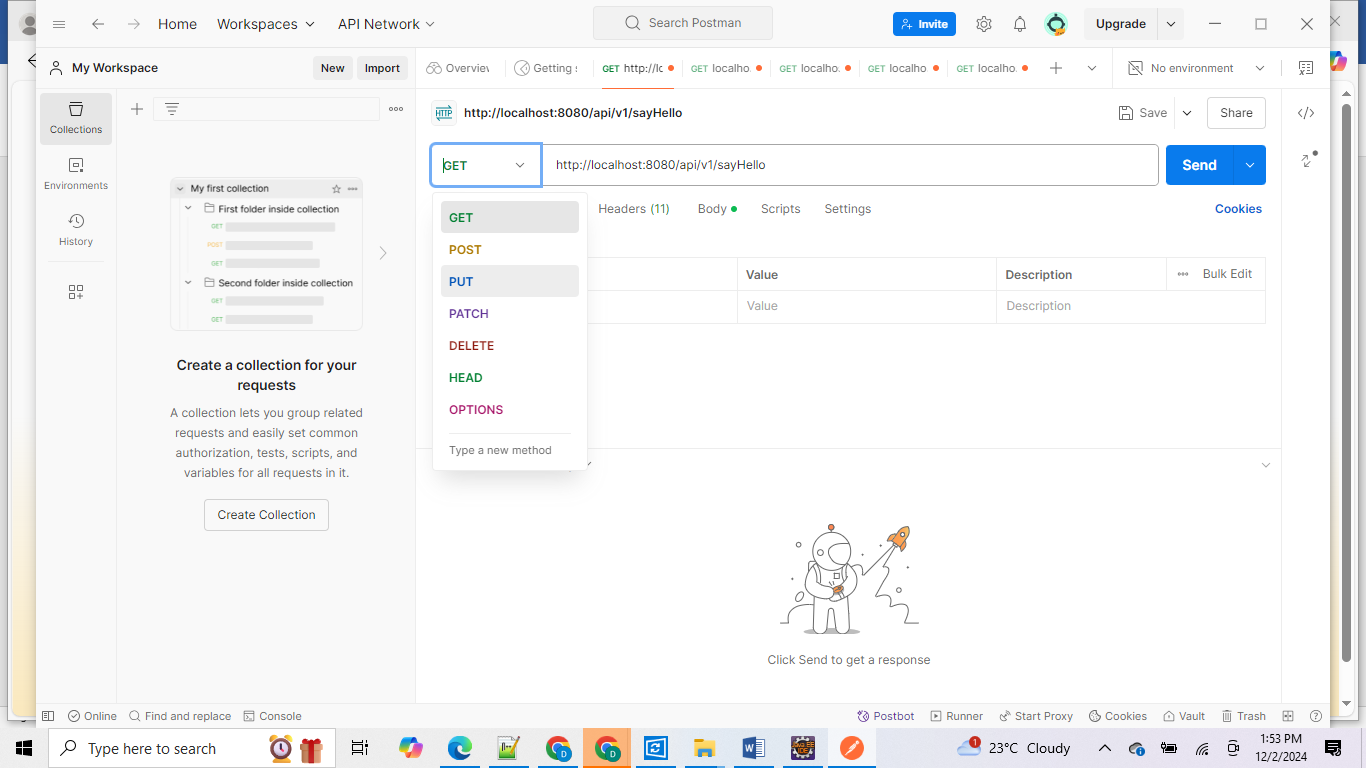
If we want to use post man we need to install post man tool first:

[Download Postman | Get Started for Free](https://www.postman.com/downloads/)



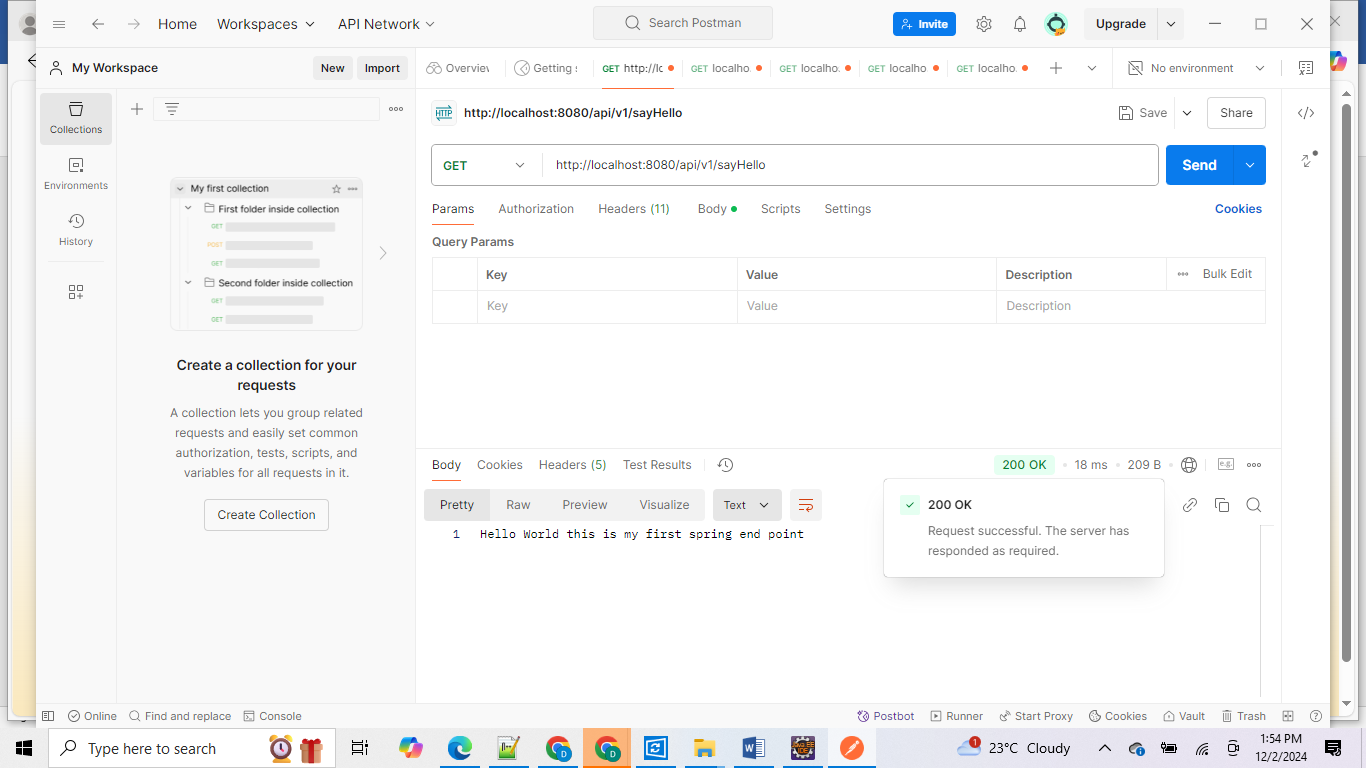
Install and login with google account then provide : user name and role and did u used postman before answers : finish set up :



Then test your api   


Dropdown we can see supporting method types from post man to test our end points:

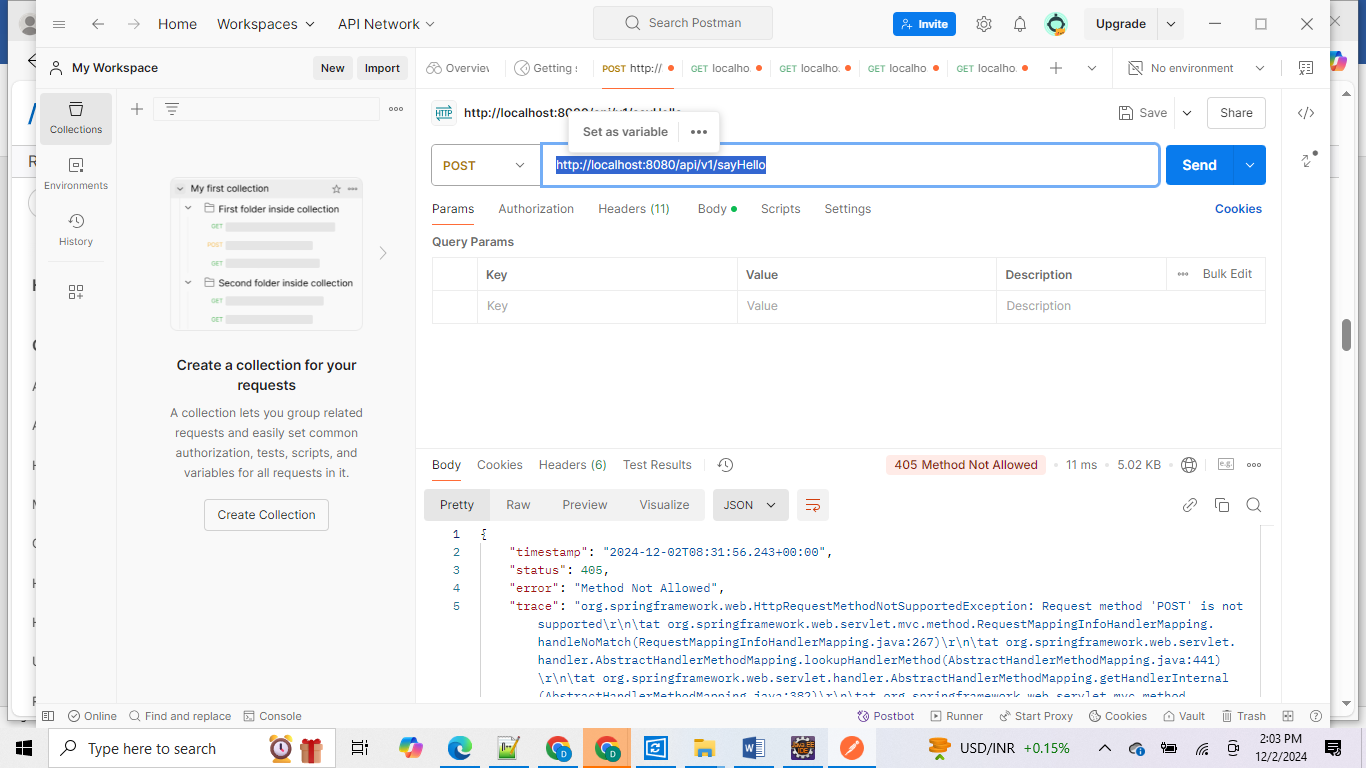
Once our request get success server will response back with HTTP Status code : 200



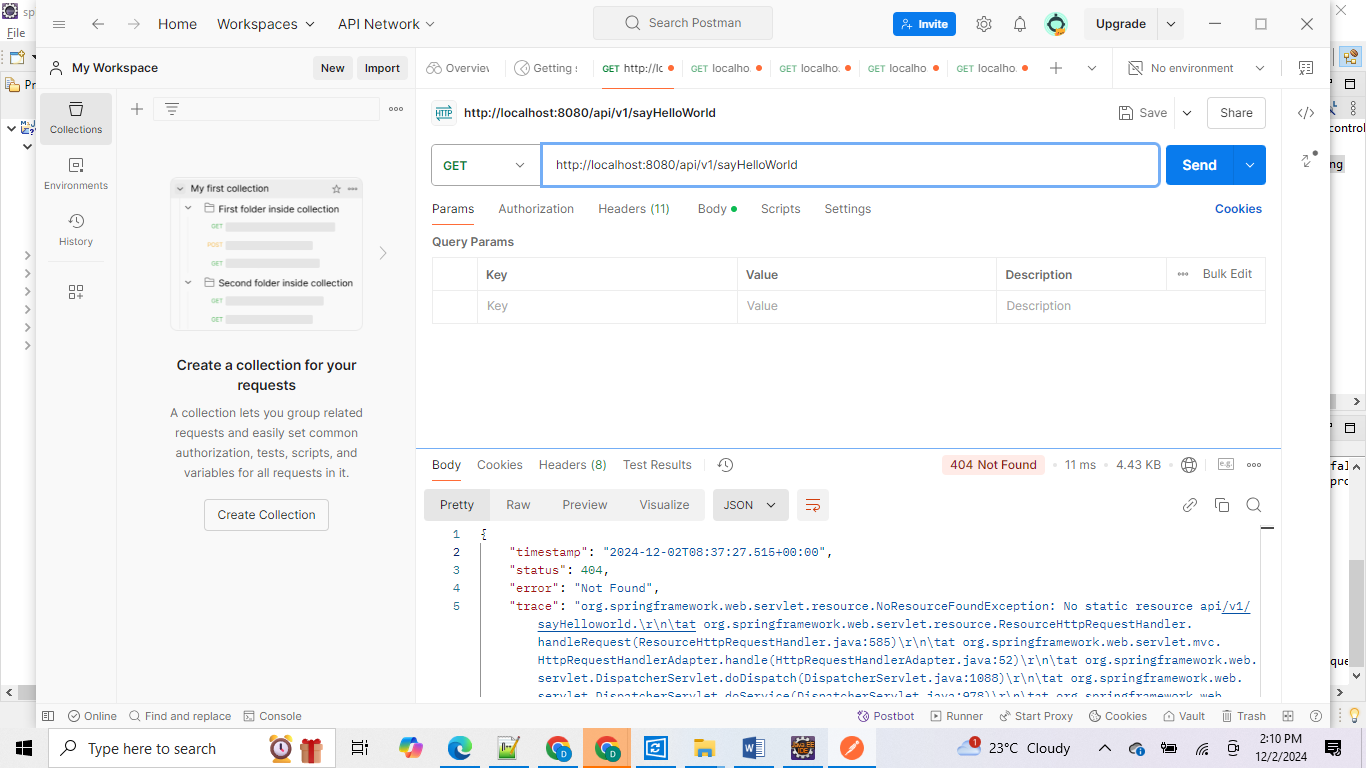
In server we have multipe range of HTTP protocal status codes :

1. [Informational responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#informational_responses) (100 – 199)
2. [Successful responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#successful_responses) (200 – 299)
3. [Redirection messages](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#redirection_messages) (300 – 399)
4. [Client error responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses) (400 – 499)
5. [Server error responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#server_error_responses) (500 – 599)

Incase our method is get, but we are calling as post : then we will get 405 error method not allowed error we will get.



If url itself wrong :  
================



[100 Continue](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/100)

This interim response indicates that the client should continue the request or ignore the response if the request is already finished.

[101 Switching Protocols](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/101)

This code is sent in response to an [Upgrade](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Upgrade) request header from the client and indicates the protocol the server is switching to.

[102 Processing](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/102) Deprecated

This code was used in [WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV) contexts to indicate that a request has been received by the server, but no status was available at the time of the response.

[103 Early Hints](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/103)

This status code is primarily intended to be used with the [Link](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link) header, letting the user agent start [preloading](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/preload) resources while the server prepares a response or [preconnect](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/preconnect) to an origin from which the page will need resources.

[**Successful responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#successful_responses)

[200 OK](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200)

The request succeeded. The result and meaning of "success" depends on the HTTP method:

* [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET): The resource has been fetched and transmitted in the message body.
* [HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD): Representation headers are included in the response without any message body.
* [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT) or [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST): The resource describing the result of the action is transmitted in the message body.
* [TRACE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/TRACE): The message body contains the request as received by the server.

[201 Created](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/201)

The request succeeded, and a new resource was created as a result. This is typically the response sent after [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST) requests, or some [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT) requests.

[202 Accepted](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202)

The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

[203 Non-Authoritative Information](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/203)

This response code means the returned metadata is not exactly the same as is available from the origin server, but is collected from a local or a third-party copy. This is mostly used for mirrors or backups of another resource. Except for that specific case, the [200 OK](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200) response is preferred to this status.

[204 No Content](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/204)

There is no content to send for this request, but the headers are useful. The user agent may update its cached headers for this resource with the new ones.

[205 Reset Content](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/205)

Tells the user agent to reset the document which sent this request.

[206 Partial Content](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/206)

This response code is used in response to a [range request](https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests) when the client has requested a part or parts of a resource.

[207 Multi-Status](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/207) ([WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV))

Conveys information about multiple resources, for situations where multiple status codes might be appropriate.

[208 Already Reported](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/208) ([WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV))

Used inside a <dav:propstat> response element to avoid repeatedly enumerating the internal members of multiple bindings to the same collection.

[226 IM Used](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/226) ([HTTP Delta encoding](https://datatracker.ietf.org/doc/html/rfc3229))

The server has fulfilled a [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET) request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.

[**Redirection messages**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#redirection_messages)

[300 Multiple Choices](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/300)

In [agent-driven content negotiation](https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation#agent-driven_negotiation), the request has more than one possible response and the user agent or user should choose one of them. There is no standardized way for clients to automatically choose one of the responses, so this is rarely used.

[301 Moved Permanently](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/301)

The URL of the requested resource has been changed permanently. The new URL is given in the response.

[302 Found](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/302)

This response code means that the URI of requested resource has been changed *temporarily*. Further changes in the URI might be made in the future, so the same URI should be used by the client in future requests.

[303 See Other](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/303)

The server sent this response to direct the client to get the requested resource at another URI with a [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET) request.

[304 Not Modified](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/304)

This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same [cached](https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching) version of the response.

[305 Use Proxy](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#305_use_proxy) Deprecated

Defined in a previous version of the HTTP specification to indicate that a requested response must be accessed by a proxy. It has been deprecated due to security concerns regarding in-band configuration of a proxy.

[306 unused](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#306_unused)

This response code is no longer used; but is reserved. It was used in a previous version of the HTTP/1.1 specification.

[307 Temporary Redirect](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/307)

The server sends this response to direct the client to get the requested resource at another URI with the same method that was used in the prior request. This has the same semantics as the 302 Found response code, with the exception that the user agent *must not* change the HTTP method used: if a [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST) was used in the first request, a POST must be used in the redirected request.

[308 Permanent Redirect](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/308)

This means that the resource is now permanently located at another URI, specified by the [Location](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Location) response header. This has the same semantics as the 301 Moved Permanently HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST) was used in the first request, a POST must be used in the second request.

[**Client error responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses)

[400 Bad Request](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/400)

The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

[401 Unauthorized](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/401)

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

[402 Payment Required](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/402)

The initial purpose of this code was for digital payment systems, however this status code is rarely used and no standard convention exists.

[403 Forbidden](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/403)

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401 Unauthorized, the client's identity is known to the server.

[404 Not Found](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404)

The server cannot find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 Forbidden to hide the existence of a resource from an unauthorized client. This response code is probably the most well known due to its frequent occurrence on the web.

[405 Method Not Allowed](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/405)

The [request method](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) is known by the server but is not supported by the target resource. For example, an API may not allow DELETE on a resource, or the TRACE method entirely.

[406 Not Acceptable](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/406)

This response is sent when the web server, after performing [server-driven content negotiation](https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation#server-driven_content_negotiation), doesn't find any content that conforms to the criteria given by the user agent.

[407 Proxy Authentication Required](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/407)

This is similar to 401 Unauthorized but authentication is needed to be done by a proxy.

[408 Request Timeout](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/408)

This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers use HTTP pre-connection mechanisms to speed up browsing. Some servers may shut down a connection without sending this message.

[409 Conflict](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/409)

This response is sent when a request conflicts with the current state of the server. In [WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV) remote web authoring, 409 responses are errors sent to the client so that a user might be able to resolve a conflict and resubmit the request.

[410 Gone](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/410)

This response is sent when the requested content has been permanently deleted from server, with no forwarding address. Clients are expected to remove their caches and links to the resource. The HTTP specification intends this status code to be used for "limited-time, promotional services". APIs should not feel compelled to indicate resources that have been deleted with this status code.

[411 Length Required](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/411)

Server rejected the request because the [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length) header field is not defined and the server requires it.

[412 Precondition Failed](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/412)

In [conditional requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Conditional_requests), the client has indicated preconditions in its headers which the server does not meet.

[413 Content Too Large](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/413)

The request body is larger than limits defined by server. The server might close the connection or return an [Retry-After](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-After) header field.

[414 URI Too Long](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/414)

The URI requested by the client is longer than the server is willing to interpret.

[415 Unsupported Media Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/415)

The media format of the requested data is not supported by the server, so the server is rejecting the request.

[416 Range Not Satisfiable](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/416)

The [ranges](https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests) specified by the Range header field in the request cannot be fulfilled. It's possible that the range is outside the size of the target resource's data.

[417 Expectation Failed](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/417)

This response code means the expectation indicated by the [Expect](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expect) request header field cannot be met by the server.

[418 I'm a teapot](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/418)

The server refuses the attempt to brew coffee with a teapot.

[421 Misdirected Request](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/421)

The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.

[422 Unprocessable Content](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422) ([WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV))

The request was well-formed but was unable to be followed due to semantic errors.

[423 Locked](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/423) ([WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV))

The resource that is being accessed is locked.

[424 Failed Dependency](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/424) ([WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV))

The request failed due to failure of a previous request.

[425 Too Early](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/425) Experimental

Indicates that the server is unwilling to risk processing a request that might be replayed.

[426 Upgrade Required](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/426)

The server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol. The server sends an [Upgrade](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Upgrade) header in a 426 response to indicate the required protocol(s).

[428 Precondition Required](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/428)

The origin server requires the request to be [conditional](https://developer.mozilla.org/en-US/docs/Web/HTTP/Conditional_requests). This response is intended to prevent the 'lost update' problem, where a client [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET)s a resource's state, modifies it and [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)s it back to the server, when meanwhile a third party has modified the state on the server, leading to a conflict.

[429 Too Many Requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/429)

The user has sent too many requests in a given amount of time ([rate limiting](https://developer.mozilla.org/en-US/docs/Glossary/Rate_limit)).

[431 Request Header Fields Too Large](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/431)

The server is unwilling to process the request because its header fields are too large. The request may be resubmitted after reducing the size of the request header fields.

[451 Unavailable For Legal Reasons](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/451)

The user agent requested a resource that cannot legally be provided, such as a web page censored by a government.

[**Server error responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#server_error_responses)

[500 Internal Server Error](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500)

The server has encountered a situation it does not know how to handle. This error is generic, indicating that the server cannot find a more appropriate 5XX status code to respond with.

[501 Not Implemented](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/501)

The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET) and [HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD).

[502 Bad Gateway](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/502)

This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.

[503 Service Unavailable](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/503)

The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-friendly page explaining the problem should be sent. This response should be used for temporary conditions and the [Retry-After](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Retry-After) HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care about the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.

[504 Gateway Timeout](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/504)

This error response is given when the server is acting as a gateway and cannot get a response in time.

[505 HTTP Version Not Supported](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/505)

The HTTP version used in the request is not supported by the server.

[506 Variant Also Negotiates](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/506)

The server has an internal configuration error: during content negotiation, the chosen variant is configured to engage in content negotiation itself, which results in circular references when creating responses.

[507 Insufficient Storage](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/507) ([WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV))

The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request.

[508 Loop Detected](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/508) ([WebDAV](https://developer.mozilla.org/en-US/docs/Glossary/WebDAV))

The server detected an infinite loop while processing the request.

[510 Not Extended](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/510)

The client request declares an HTTP Extension ([RFC 2774](https://datatracker.ietf.org/doc/html/rfc2774)) that should be used to process the request, but the extension is not supported.

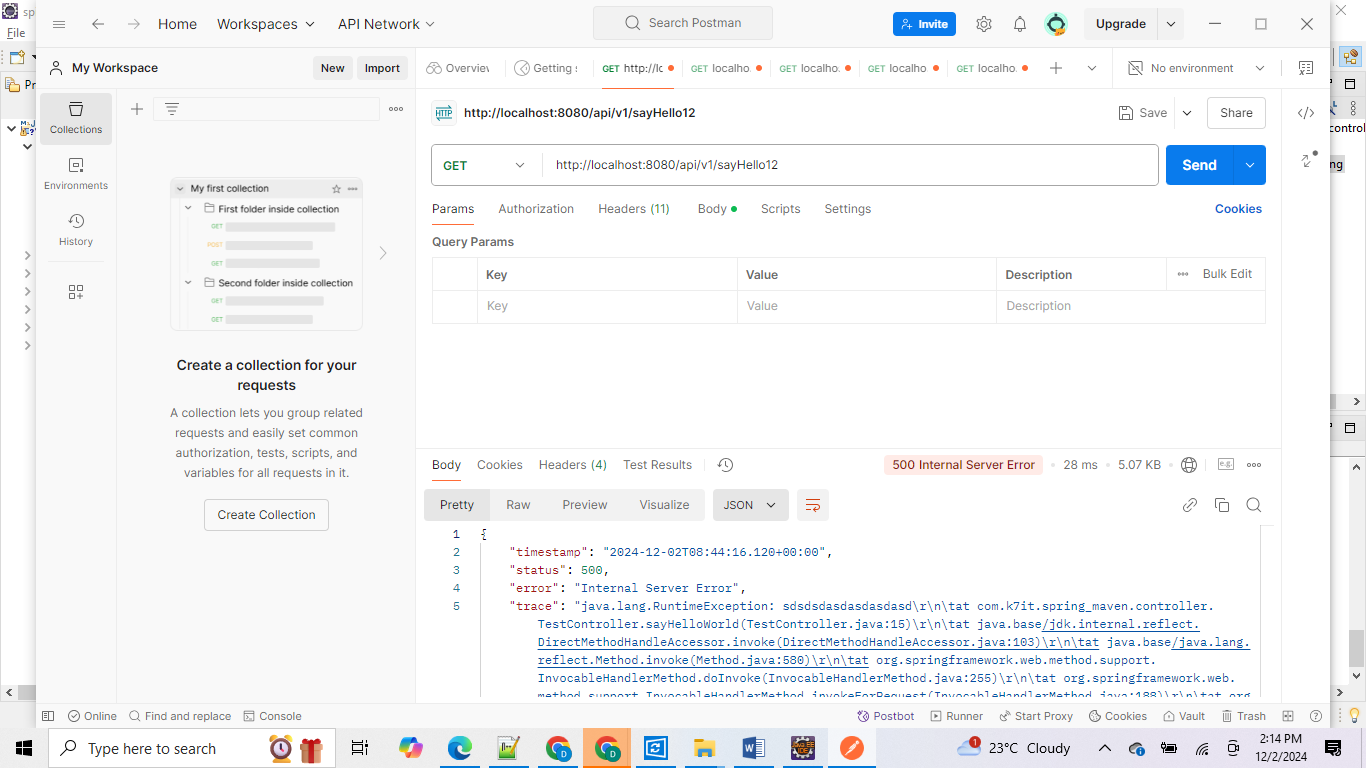
[511 Network Authentication Required](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/511)

Indicates that the client needs to authenticate to gain network access.

Important codes:

200, 201, 203, 400,404,405,500,501

In case if we send any exceptions from back end then we will 500 error :



**package** com.k7it.spring\_maven.controller;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.RequestMapping;

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

@RestController

@RequestMapping("/api/v1")

**public** **class** TestController {

@GetMapping("/sayHello12")// final end point class end point + method end point = "/api/v1/sayHello"

**public** String sayHelloWorld() {

**if**(**true**) {

**throw** **new** RuntimeException("sdsdsdasdasdasdasd");

}

**return** "Hello World this is my first spring end point";

}

}

Local host DNS mapping will do inside our system under hosts file in this location   
**C:\Windows\system32\drivers\etc**# Copyright (c) 1993-2009 Microsoft Corp.

#

# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.

#

# This file contains the mappings of IP addresses to host names. Each

# entry should be kept on an individual line. The IP address should

# be placed in the first column followed by the corresponding host name.

# The IP address and the host name should be separated by at least one

# space.

#

# Additionally, comments (such as these) may be inserted on individual

# lines or following the machine name denoted by a '#' symbol.

#

# For example:

#

# 102.54.94.97 rhino.acme.com # source server

# 38.25.63.10 x.acme.com # x client host

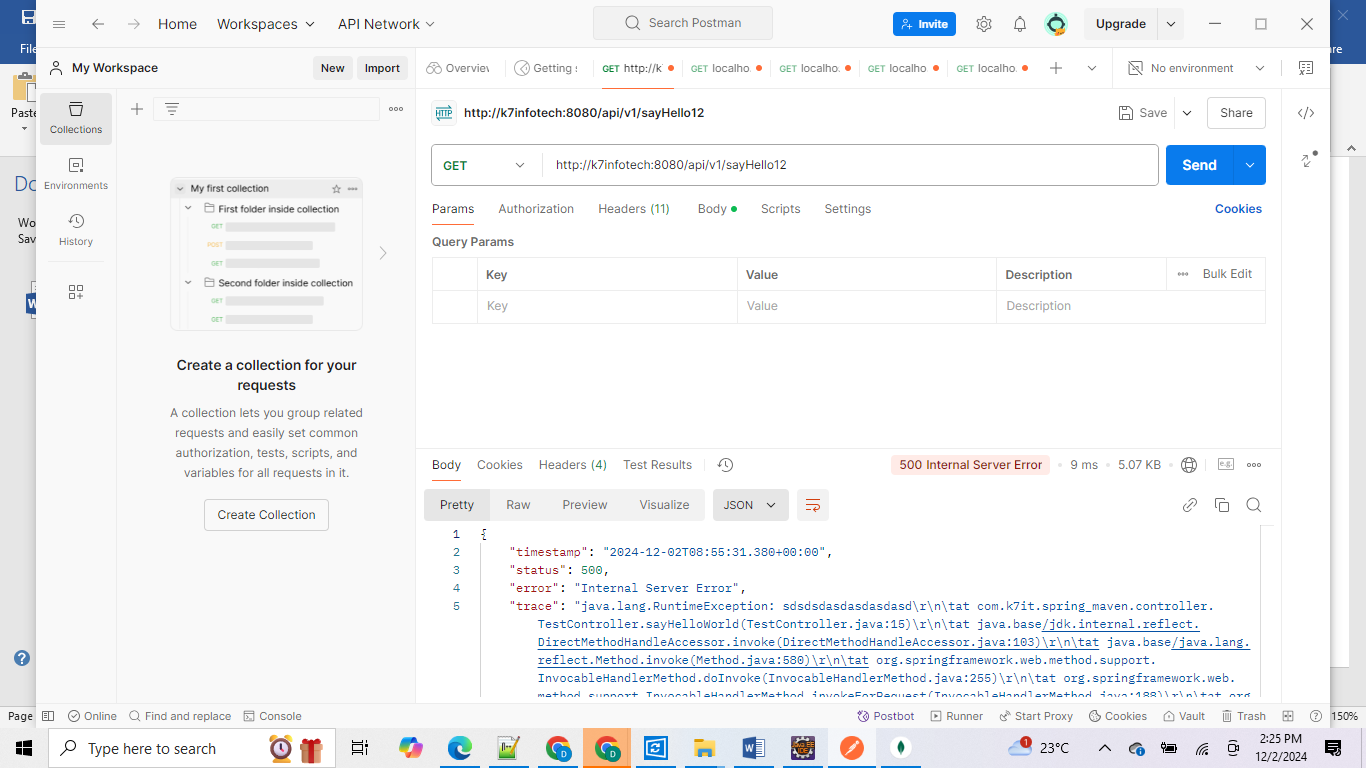
# localhost name resolution is handled within DNS itself.

# 127.0.0.1 localhost

# ::1 localhost

127.0.0.1 k7infotech

127.0.0.1 localhost



1. How to install Lombok plugin in eclipse :  
   ==============================  
   **Download Lombok**: Visit the Lombok website and download the lombok.jar file.  
   [Download](https://projectlombok.org/download?form=MG0AV3)
2. **Run the Lombok Installer**: Open a terminal or command prompt and run the following command:
3. java -jar lombok.jar

This will open the Lombok installer.

1. **Install Lombok**: In the installer, click the "Install/Update" button. The installer will automatically detect your Eclipse installation1. If it doesn't, you can manually specify the Eclipse installation directory.
2. **Restart Eclipse**: After the installation is complete, restart Eclipse.
3. **Verify Installation**: To check if Lombok is installed correctly, go to Help > About Eclipse. You should see the Lombok version listed at the bottom of the dialog1.
4. **Enable Annotation Processing**: Go to Window > Preferences > Java > Compiler > Annotation Processing and make sure "Enable annotation processing" is checked.
5. **Add Lombok to Project Classpath**: Add the Lombok dependency to your project's build tool (e.g., Maven or Gradle). For Maven, add the following dependency:

xml

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.30</version>

<scope>provided</scope>

</dependency>

For Gradle, add:

groovy

dependencies {

compileOnly 'org.projectlombok:lombok:1.18.30'

annotationProcessor 'org.projectlombok:lombok:1.18.30'

}

That's it! You should now be able to use Lombok in your Eclipse projects. If you run into

Example 2:

**package** com.k7it.sms.controller;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

**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.RequestParam;

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

**import** com.k7it.sms.model.Student;

@RestController

@RequestMapping("/api/v1")

**public** **class** StudentController {

@GetMapping("/test")

**public** String test() {

**return** "Hello World";

}

@PostMapping("/createStudent")

**public** String createStudent( @RequestBody Student student ) {

**return** student.toString();

}

@PostMapping("/addStudent")

**public** String addStudent(@RequestParam **int** id, @RequestParam String name, @RequestParam String branch, @RequestParam **int** marks, @RequestParam String address) {

**return** **new** Student(id, name, branch, marks, address).toString();

}

@GetMapping("/add")

**public** **int** add(@RequestParam **int** n1, @RequestParam **int** n2) {

**return** n1+n2;

}

@GetMapping("/getStudent/{id}")

**public** String getStudent( @PathVariable **int** id) {

**return** "Student details for id:"+ id+" is:"+ **new** Student(101,"k7it","cse",99,"Bangalore").toString();

}

**public** **static** **void** main(String[] args) {

StudentController sc = **new** StudentController();

Student s1 = **new** Student(101,"k7it","cse",99,"bangalore");

sc.createStudent(s1);

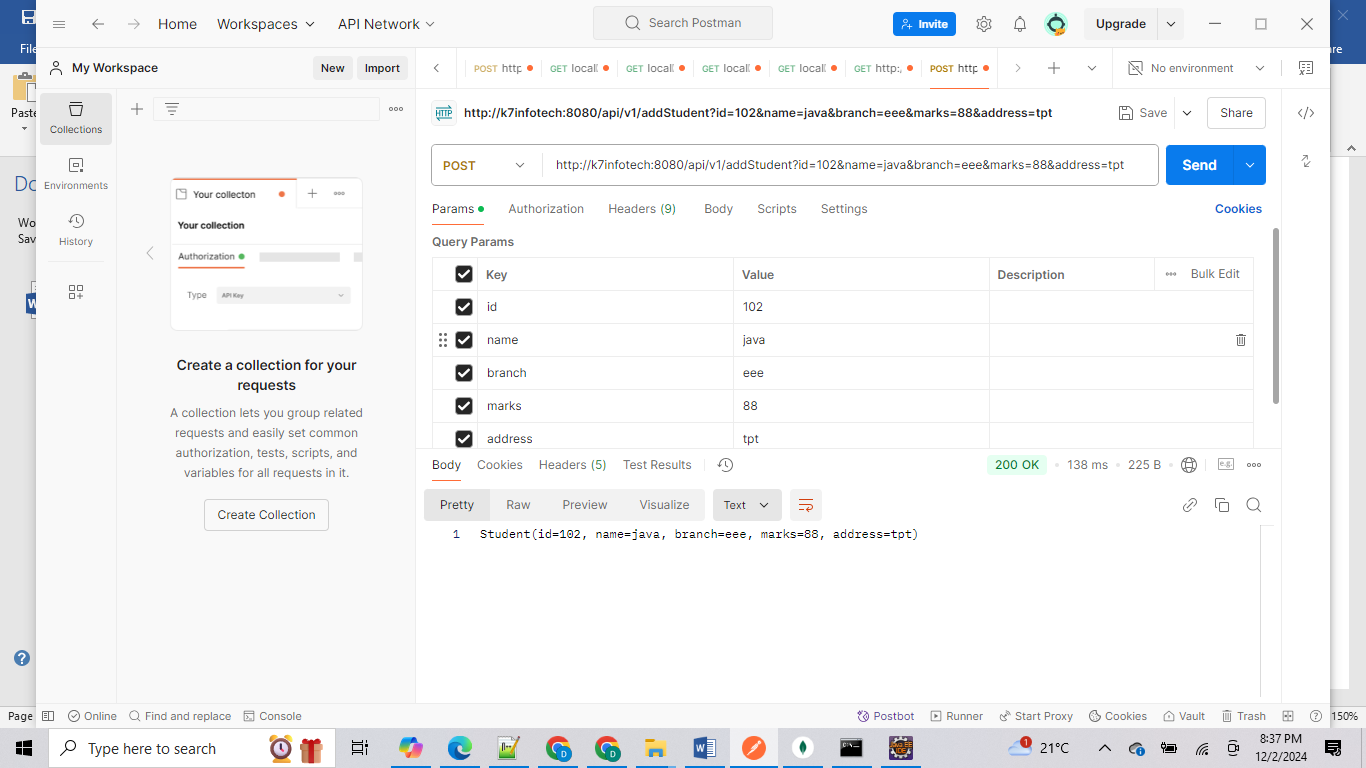
// in case of java we can specify method params by seeing method signature. how to specifiy about our method input params to user or front users

// we can specifies about input paramt to end user through spring boot in 3 ways

/\*

1. @RequestParam : this annotation we will use mainly for if we want to pass primitive attributes like int, float, double, string etc.

in case of REquestparam case data has to pass as part of Query String from the ui or post man tool or under params tabs we should add our param names as keys and corresponding values



2. @PathParam or @PathVariable : this is annotation we will use to specify to user if want tao take input params as part of the our method URI

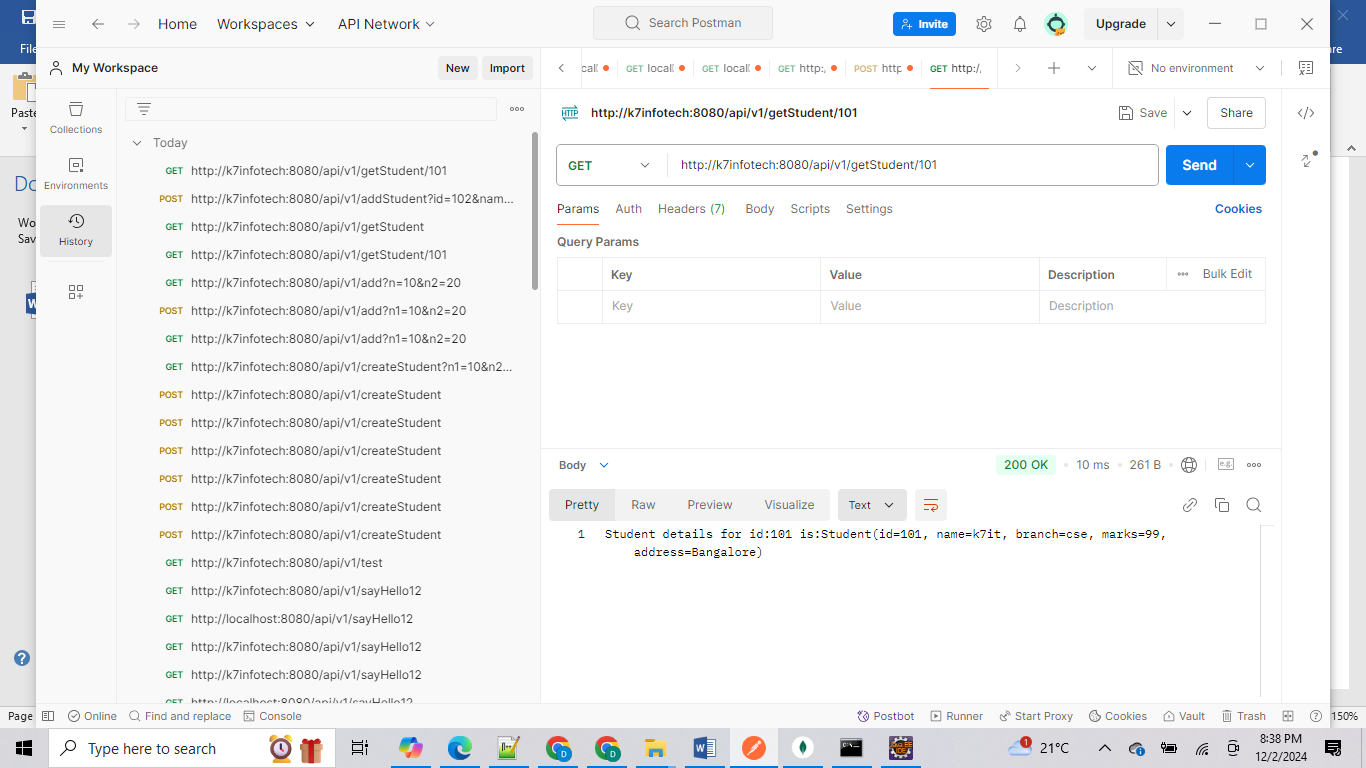
@GostMapping("/student/{id}")

public Student getStudentDetails(@PathVariable int id){

}

if user wan to call this method he should use the uri like bellow

http://localhost:8080/api/v1/student/101 -> 101 will go value for id field in method param



3. @RequestBody : by using this annotation we can accept derived data type input params like Student, or Person or Addrss or Animal

for this Requestbody case or derived attribute case front end user should pass values as JSON format like key and value pairs

for example if we want to pass value for Student object use each attrubute as key for the JSON object and pass value for each key based on corresponding

attribute data type.

{

"sId":101,

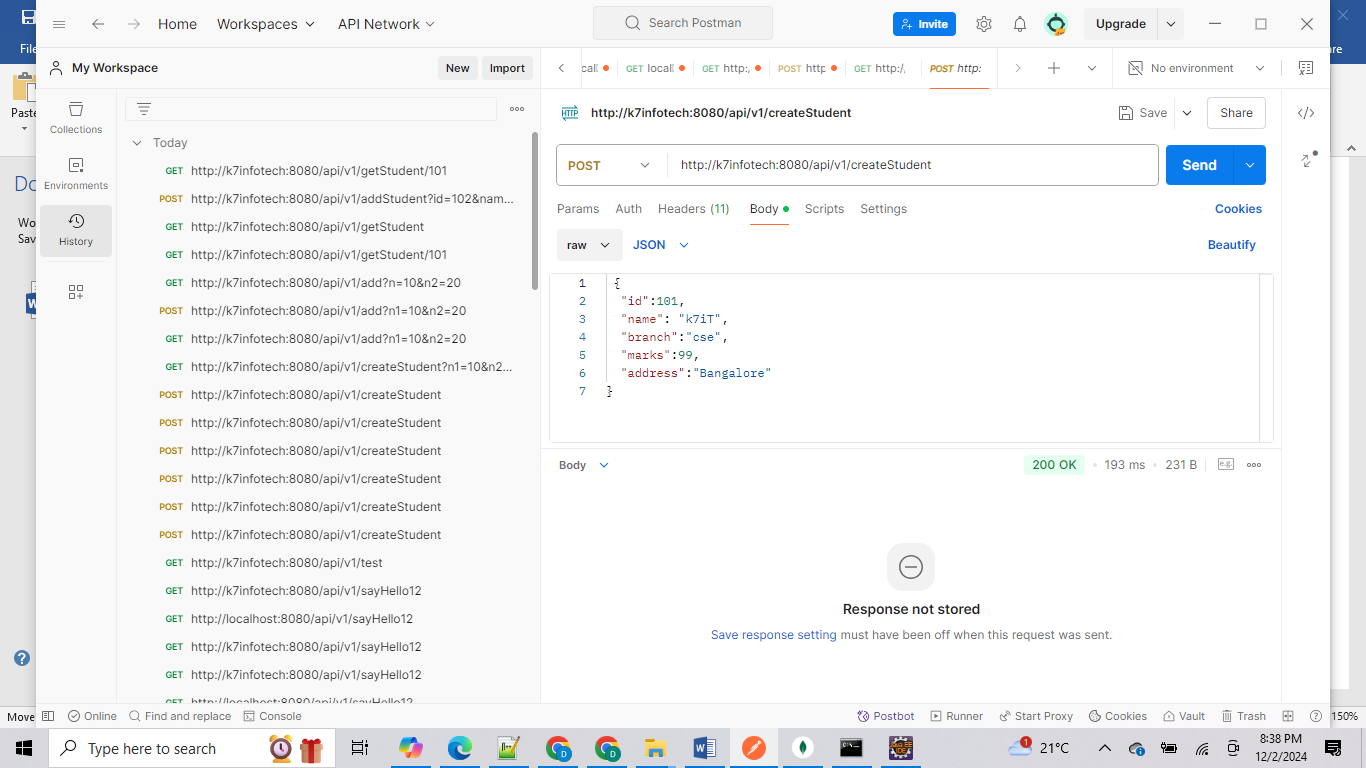
"sName": "k7iT",

"branch":"cse",

"marks":99,

"address":"Bangalore"

}



\*/

}

}

In case if we want to pass multiple derived objects as input params in Request body :  
  
then create separate class with those two derived attributes and use that new class as Requestbody attribute:

@PostMapping("/studentAddress")

**public** StudentAddress insertStudentWithAddress(@RequestBody StudentAddress studentAddress) {

Student student = studentAddress.getStudent();

Address address = studentAddress.getAddress();

**return** studentAddress;

}

**package** com.k7it.sms.model;

**import** lombok.AllArgsConstructor;

**import** lombok.Data;

**import** lombok.NoArgsConstructor;

**import** lombok.ToString;

@Data

@ToString

@AllArgsConstructor

@NoArgsConstructor

**public** **class** StudentAddress {

**private** Student student;

**private** Address address;

}

**package** com.k7it.sms.model;

**import** lombok.AllArgsConstructor;

**import** lombok.Data;

**import** lombok.NoArgsConstructor;

**import** lombok.ToString;

@Data

@AllArgsConstructor

@NoArgsConstructor

@ToString

**public** **class** Address {

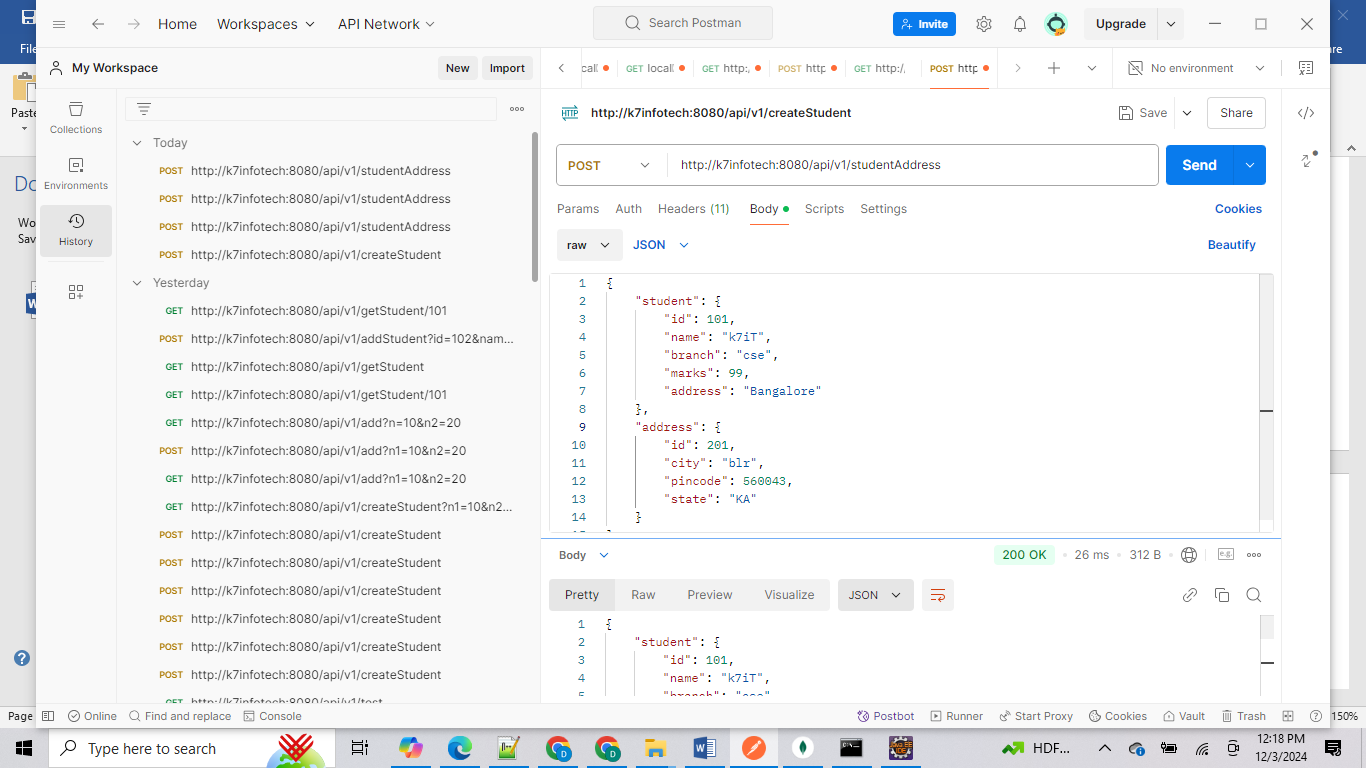
**int** id;

String city;

**int** pincode;

String state;

}



Input req body:  
  
{

    "student": {

        "id": 101,

        "name": "k7iT",

        "branch": "cse",

        "marks": 99,

        "address": "Bangalore"

    },

    "address": {

        "id": 201,

        "city": "blr",

        "pincode": 560043,

        "state": "KA"

    }

}

Output also we will get same format.

{

    "student": {

        "id": 101,

        "name": "k7iT",

        "branch": "cse",

        "marks": 99,

        "address": "Bangalore"

    },

    "address": {

        "id": 201,

        "city": "blr",

        "pincode": 560043,

        "state": "KA"

    }

}

Let start interaction with Database: if we want to interact with database Repository layer help : it will act like mediator between java to database. For this we need to add corresponding dependency based on data type if SQL or Oracle use Spring data JPA dependency in case of MongoDB we should use Spring data MongoDB dependency.

Let start with mongodb : add in pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

Then create Repository interface making sub class of MongoRepository<EntityClssname, Id field data type> with two generics input params.

**package** com.k7it.sms.repo;

**import** org.springframework.data.mongodb.repository.MongoRepository;

**import** com.k7it.sms.model.Student;

**public** **interface** StudentRepo **extends** MongoRepository<Student, Integer> {

}

Map the Student Enity class with database Table using Spring data JPA annotations

Like in case of mysql or oracle for our pojo class we should use @Entity annotation

In case of MongoDB we should use @Document annotation

Use Repo class to save this entity class object into database either from Controller class or Service class. Inside Repo class we have predefined database related operations methods like save or update or delete or findByID() etc..

Curd operations:

**package** com.k7it.sms.controller;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.web.bind.annotation.DeleteMapping;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.PutMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

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

**import** com.k7it.sms.model.Person;

**import** com.k7it.sms.service.PersonService;

@RestController

@RequestMapping("api/v1")

**public** **class** PersonController {

@Autowired

PersonService personService;

@PostMapping("/person")

**public** Person addPerson(@RequestBody Person person) {

**return** personService.addPerson(person);

}

@PutMapping("/person/{id}")

**public** Person updatePerson(@PathVariable **int** id,@RequestBody Person person) {

**return** personService.updatePerson(id,person);

}

@DeleteMapping("/person/{id}")

**public** Person deletePerson(@PathVariable **int** id) {

**return** personService.deletePerson(id);

}

@GetMapping("/person/{id}")

**public** Person getPersonDetails(@PathVariable **int** id) {

**return** personService.getPersonDetails(id);

}

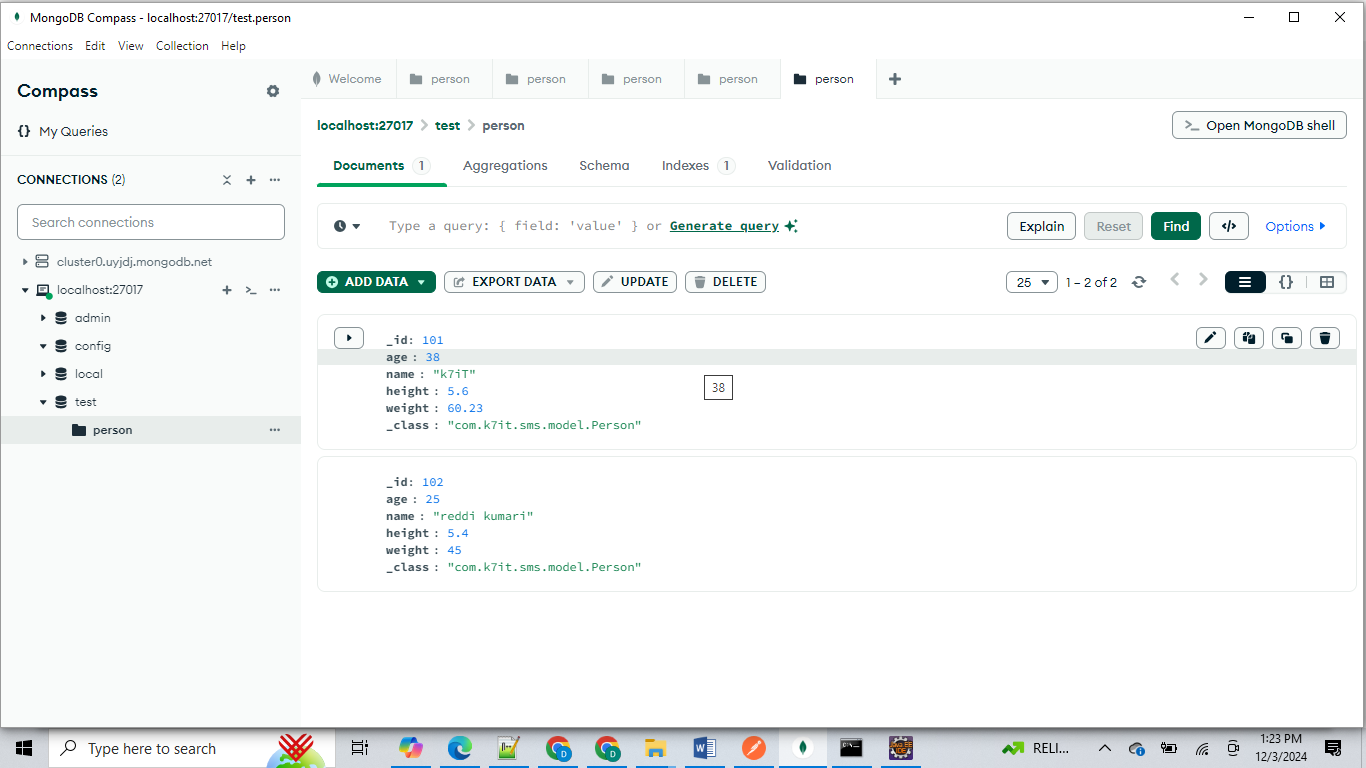
@GetMapping("/persons")

**public** List<Person> getAllPerons(){

**return** personService.getAllPersons();

}

}



If we want to see the data or Collection in mongoDB , at least one record should insert through our api. Then automatically it will create database name as Test with default db name and inside that it will create collection with the name of our entity class i.e Person and it will insert the data.

Here till now we didn’t given any configuration info for our mongodb, even though not provided our spring boot project automatically connected with our local mongodb internally. In case mongodb is not running in our system if we want to connect databse in cloud env or other system location then we should provide mongodb connection URI string else it work.

How to give mongodb conection uri string:

Open Application.properties file :

Add bellow key and value

spring.application.name=sms

server.port=8090

spring.data.mongodb.uri=mongodb://localhost:27017/sms

by default it will take it as test database .

how to copy connection String URI: go to compass-> select localhost server -> click on three dots menu -> click copy connection String ->

mongodb://localhost:27017 after this this append your database name which you want to use   
  
mongodb://localhost:27017/sms -> here sms is my db name.

server.port= 8090 -> by using this key we can change our server port number which one we want instead of using default port number as 8080.

MVC: 3tier architecture (or) three tier architecture

Model: the class which is using to transfer the data from java to Database

Controller : the class which is using to interact between UI to Java with the help of end points or API’s.

View: the files which are representing front end code , this file act like output from java to UI to render the response data into User .

DTO: Data transfer object : this is class also act like same as Model class which is also used to transfer the data from one layer to another layer it can be a ui to java

Or java to java or java to database.

Spring JPA:

Spring JPA is mainly used to connect from java to Database it works same as hibernate framework. It is also called ORM : Object Relational Model.

Important protocols :

### Networking Protocols

1. **TCP/IP (Transmission Control Protocol/Internet Protocol)**: The fundamental suite for data exchange over the internet.
2. **UDP (User Datagram Protocol)**: A simpler, connectionless protocol used in applications where speed is more critical than reliability.
3. **ICMP (Internet Control Message Protocol)**: Used for error messages and operational information.
4. **ARP (Address Resolution Protocol)**: Resolves IP addresses to MAC addresses.

### Web Protocols

1. **HTTP (Hypertext Transfer Protocol)**: The foundation of data communication for the web.
2. **HTTPS (Hypertext Transfer Protocol Secure)**: HTTP with encryption for secure communication.
3. **FTP (File Transfer Protocol)**: Used for transferring files between a client and a server.
4. **SFTP (SSH File Transfer Protocol)**: FTP over SSH for secure file transfers.

### Email Protocols

1. **SMTP (Simple Mail Transfer Protocol)**: Used for sending emails.
2. **POP3 (Post Office Protocol version 3)**: Used by email clients to retrieve emails from a server.
3. **IMAP (Internet Message Access Protocol)**: Another protocol for retrieving emails, allowing more complex mail management.

### Security Protocols

1. **SSL/TLS (Secure Sockets Layer/Transport Layer Security)**: Cryptographic protocols for secure communications over a network.
2. **IPSec (Internet Protocol Security)**: A suite of protocols for securing IP communications by authenticating and encrypting each IP packet.

### Wireless Protocols

1. **Wi-Fi (Wireless Fidelity)**: A family of wireless networking protocols based on the IEEE 802.11 standards.
2. **Bluetooth**: A wireless technology standard for exchanging data over short distances.

### File Transfer Protocols

1. **FTPS (FTP Secure)**: An extension to FTP with added support for encryption.
2. **TFTP (Trivial File Transfer Protocol)**: A simpler version of FTP, often used for transferring small amounts of data.

### Streaming Protocols

1. **RTSP (Real-Time Streaming Protocol)**: A network control protocol designed for use in entertainment and communications systems to control streaming media servers.
2. **RTP (Real-Time Transport Protocol)**: A protocol for delivering audio and video over IP networks.

### Internet of Things (IoT) Protocols

1. **MQTT (Message Queuing Telemetry Transport)**: A lightweight messaging protocol for small sensors and mobile devices.
2. **CoAP (Constrained Application Protocol)**: A specialized web transfer protocol for use with constrained nodes and constrained networks in the IoT.

### Data Link Layer Protocols

1. **Ethernet**: A family of networking technologies for local area networks (LANs).
2. **PPP (Point-to-Point Protocol)**: Used to establish a direct connection between two networking nodes.

### Others

1. **DNS (Domain Name System)**: Translates domain names to IP addresses.
2. **DHCP (Dynamic Host Configuration Protocol)**: Assigns IP addresses to devices on a network automatically.

Similar to this protocols there is ORM tool to communicate between Java to database. Older days we are using JDBC(java data base connectivity) for java to Relational database communication purpose.

Later Hibernate framework came into market to simplify the Database connectivity from java. Java developers will use hibernate from java but hibernate will use jdbc internally to communicate java to DB.

Now Spring boot came up with JPA framework , so we will use JPA framework to communicate spring based application to database, but JPA will use internally hibernate and jdbc concepts only.

To communicate java to database main point of contact from java using jpa is model class or entity class and Repository class

Here

Model : Model class will have information or mapping details about java class to table and java class attributes to table columns and also mapping or relationships from one table to another table

Repository : this class will maintain the connectivity between java to database , he will only transfer request from java to database and response from database to java.

In older days if we want to connect from java to database we are using jdbc driver

table with JDBC drivers, Maven dependencies, and application.properties configurations for various databases, including Spring Data JPA dependencies.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Database** | **JDBC Driver Class** | **Maven Dependency** | **Database URL Example** | **application.properties Configuration** |
| **MySQL** | com.mysql.jdbc.Driver | xml <dependency> <groupId>com.mysql</groupId> <artifactId>mysql-connector-j</artifactId> <version>8.0.30</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:mysql://localhost:3306/mydatabase | properties spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=com.mysql.jdbc.Driver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **Oracle** | oracle.jdbc.driver.OracleDriver | xml <dependency> <groupId>com.oracle.ojdbc</groupId> <artifactId>ojdbc8</artifactId> <version>21.3.0.0</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:oracle:thin:@localhost:1521:xe | properties spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **PostgreSQL** | org.postgresql.Driver | xml <dependency> <groupId>org.postgresql</groupId> <artifactId>postgresql</artifactId> <version>42.4.0</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:postgresql://localhost:5432/mydatabase | properties spring.datasource.url=jdbc:postgresql://localhost:5432/mydatabase spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=org.postgresql.Driver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **SQL Server** | com.microsoft.sqlserver.jdbc.SQLServerDriver | xml <dependency> <groupId>com.microsoft.sqlserver</groupId> <artifactId>mssql-jdbc</artifactId> <version>12.8.1.jre11</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:sqlserver://localhost:1433;databaseName=mydatabase | properties spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=mydatabase spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **DB2** | com.ibm.db2.jcc.DB2Driver | xml <dependency> <groupId>com.ibm.db2</groupId> <artifactId>jcc</artifactId> <version>11.5.8.0</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:db2://localhost:50000/mydatabase | properties spring.datasource.url=jdbc:db2://localhost:50000/mydatabase spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=com.ibm.db2.jcc.DB2Driver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **Sybase** | com.sybase.jdbc.SybDriver | xml <dependency> <groupId>net.sourceforge.jtds</groupId> <artifactId>jtds</artifactId> <version>1.3.1</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:sybase:Tds://localhost:4100/mydatabase | properties spring.datasource.url=jdbc:sybase:Tds://localhost:4100/mydatabase spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=com.sybase.jdbc.SybDriver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **SQLite** | org.sqlite.JDBC | xml <dependency> <groupId>org.xerial</groupId> <artifactId>sqlite-jdbc</artifactId> <version>3.47.1</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:sqlite:/path/to/database/file | properties spring.datasource.url=jdbc:sqlite:/path/to/database/file spring.datasource.driver-class-name=org.sqlite.JDBC spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **MongoDB** | com.mongodb.MongoClient | xml <dependency> <groupId>org.mongodb</groupId> <artifactId>mongo-java-driver</artifactId> <version>4.5.0</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-mongodb</artifactId> </dependency> | mongodb://localhost:27017/mydatabase | properties spring.data.mongodb.uri=mongodb://localhost:27017/mydatabase logging.level.org.springframework.data.mongodb.core.MongoTemplate=DEBUG logging.level.org.mongodb.driver=DEBUG |
| **MariaDB** | org.mariadb.jdbc.Driver | xml <dependency> <groupId>org.mariadb.jdbc</groupId> <artifactId>mariadb-java-client</artifactId> <version>2.7.4</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:mariadb://localhost:3306/mydatabase | properties spring.datasource.url=jdbc:mariadb://localhost:3306/mydatabase spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=org.mariadb.jdbc.Driver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |
| **H2** | org.h2.Driver | xml <dependency> <groupId>com.h2database</groupId> <artifactId>h2</artifactId> <version>2.1.214</version> </dependency> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | jdbc:h2:~/test | properties spring.datasource.url=jdbc:h2:~/test spring.datasource.driver-class-name=org.h2.Driver spring.datasource.username=sa spring.datasource.password=password spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true |

This table provides you info about all databases dependency jars and connection url and user name and password and driver classes info :

Note : for **mongodb we need only connection URI**

but all other databases we need **connection url and also user name , password and driver class details** total 4 details min we need configure in application.properties file

JDBC steps :

1. Load driver class

try {

Class.forName("com.mysql.jdbc.Driver"); // For MySQL

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

1. Establish a Connection

String url = "jdbc:mysql://localhost:3306/mydatabase";

String username= "myuser";

String password = "mypassword";

try {

Connection connection = DriverManager.getConnection(url, user, password);

System.out.println("Connection established successfully.");

} catch (SQLException e) {

e.printStackTrace();

}

1. Create a Statement or PreparedStatement : this is the class which is perfroming databse request and response activities .

try {

Statement statement = connection.createStatement();

} catch (SQLException e) {

e.printStackTrace();

}

1. Execute user queries like insert or update or select or delete

try {

String query = "SELECT \* FROM mytable";

ResultSet resultSet = statement.executeQuery(query);

while (resultSet.next()) {

System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"));

}

} catch (SQLException e) {

e.printStackTrace();

}

Here ResultSet class act like response from database or return value from statement.

1. Use the response or resultSet

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class JdbcExample {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/mydatabase";

String user = "myuser";

String password = "mypassword";

Connection connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

// Load and register JDBC driver

Class.forName("com.mysql.jdbc.Driver");

// Establish connection

connection = DriverManager.getConnection(url, user, password);

System.out.println("Connection established successfully.");

// Create a statement

statement = connection.createStatement();

// Execute a query

String query = "SELECT id, name, age FROM mytable";

resultSet = statement.executeQuery(query);

// Process the result set

while (resultSet.next()) {

// Retrieve data by column name

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

int age = resultSet.getInt("age");

// Display the data

System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age);

}

} catch (ClassNotFoundException e) {

e.printStackTrace();

} catch (SQLException e) {

e.printStackTrace();

} finally {

// Close the result set, statement, and connection

try {

if (resultSet != null) resultSet.close();

if (statement != null) statement.close();

if (connection != null) connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

1. Close all resource objects in reverse order

try {

resultSet.close();

statement.close();

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

JPA:

JPA is advanced concepts to JDBC from Spring or SpringBoot . instead of doing such a long process and execute native sql queries from java we can use java class objects. If we use jpa we can perform db operations even though if we don’t have any knowledge about database queries with the help repository class and Entity classes .

Application.properties : it will contain all connection config info for databases

Driver, url, username, password. For each value there is sperate key in properties file we need to set those keys based on database like Mysql, oracle, h2, db2, postgreySQL etc.

Model class: this will have info about java class to database table mapping details

Like table name and column name mapping with our class and attributes based on SQL (Relational based )and No SQL these maping annotations may vary.

In model class if we want to map our model class MongoDb we should use @Document (Collection=”collection\_name”)  
but in case SQL databases we should use @Entity(table = “tableName”)

Here Table or Collection attributes we can use only if we dont want to use same class name as table names in database .

Repository class: this class will make connection from java to database and he will only execute all our db requests and he will provide response back from DB.

Here making our Repository class as sub class to JPA resposity class may change based SQL and NoSQL bases

SQL : JPARepository

NOSQL : MongoDB : MongoDriver

SQL VS NOSQL JPA :

table that outlines the differences between SQL and NoSQL databases in the context of Spring Data JPA, including related properties, entity classes, repository classes, and more.

| Aspect | **SQL (Relational Database)** | **NoSQL (Document Database - MongoDB)** |
| --- | --- | --- |
| Database Type | Relational (e.g., MySQL, PostgreSQL, Oracle) | Document-Oriented (e.g., MongoDB) |
| Data Model | Tables with rows and columns | Collections of documents |
| Schema | Fixed schema | Dynamic schema (schema-less) |
| Primary Key | Typically an auto-incrementing integer or UUID | Typically an ObjectId generated by MongoDB |
| Entity Class | Annotated with @Entity and @Table | Annotated with @Document |
| Repository | Extends JpaRepository or CrudRepository | Extends MongoRepository or CrudRepository |
| SQL Example | sql CREATE TABLE person (id INT PRIMARY KEY, name VARCHAR(255), age INT); | json { "\_id": ObjectId("60c72b2f4f1a2c6d088b4567"), "name": "John", "age": 30 } |
| Spring Data Dependency | xml <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency> | xml <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-data-mongodb</artifactId> </dependency> |
| **application.properties Configuration** | properties spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase spring.datasource.username=myuser spring.datasource.password=mypassword spring.datasource.driver-class-name=com.mysql.jdbc.Driver spring.jpa.hibernate.ddl-auto=update  spring.jpa.show-sql=true | properties spring.data.mongodb.uri=mongodb://localhost:27017/mydatabase logging.level.org.springframework.data.mongodb.core.MongoTemplate=DEBUG logging.level.org.mongodb.driver=DEBUG |
| Entity Class Example | java import javax.persistence.Entity; import javax.persistence.Id;   @Entity  public class Person {  @Id  private int id; private String name;  private int age;  // Getters and setters } | java import org.springframework.data.annotation.Id; import org.springframework.data.mongodb.core.mapping.Document; @Document(collection = "person") public class Person {  @Id  private String id;  private String name;  private int age;  // Getters and setters  } |
| Repository Interface Example | java import org.springframework.data.jpa.repository.JpaRepository; public interface PersonRepository extends JpaRepository<Person, Integer> {} | java import org.springframework.data.mongodb.repository.MongoRepository; public interface PersonRepository extends MongoRepository<Person, String> {} |

### Explanation

#### **SQL (Relational Database)**

* **Entity Class**: Uses @Entity and @Table annotations.
* **Primary Key**: Defined with @Id and usually auto-incremented.
* **Repository**: Extends JpaRepository or CrudRepository.
* **Properties**: Configurations like spring.datasource.url, spring.jpa.hibernate.ddl-auto, and spring.jpa.show-sql.

#### **NoSQL (Document Database - MongoDB)**

* **Entity Class**: Uses @Document annotation.
* **Primary Key**: Defined with @Id and typically an ObjectId.
* **Repository**: Extends MongoRepository or CrudRepository.
* **Properties**: Configurations like spring.data.mongodb.uri and logging levels for debugging.

This table highlights the key differences and provides example configurations and code snippets for both SQL and NoSQL databases in the context of Spring Data JPA

Mappings in JPA or ORM :  
===================  
In Database we are having 4 types mapping based their relationships

1. One to One
2. One to Many
3. Many to One
4. Many to Many  
     
   for example

(One-to-One) -> one student can have one address

Student -> Address address -> inside Student class we have one Address   
 Address -> Student address;  
  
Class Student {

// Student details attribtes   
 Address address;

}

Address -> no need to mention anything related to Student

(One-to-Many) -> Multiple projects into Single student

Student -> List<Project> projects;  
 Project -> Student student;

class Student{  
  
 List<Project> projects:

}

Project -> no need to mention anything about student or Student student ;

(Many-to-One)

Student -> School (Many-to-One) --> multiple students into one school

Student -> School school

School -> List<Student> students

(Many to Many )

Student -> List<Course> //{java,mongodb,sql,git,junits,log4j,html}

Course -> List<Student> //{kesav,gnana,tharun,redid,prasad,kumar,manish.sai,babu,thulasi}