**1. Explain in Detail about HTTP Request and Response**

🔹 HTTP Request

* A client sends a request to a server using the HTTP protocol.
* Format:

CODE:

GET /path/resource HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0

Content-Type: application/json

* Key Components:
  + Request URL: Location of resource (e.g., http://example.com/api/users)
  + Request Method: e.g., GET, POST, PUT, DELETE
  + Headers: Meta info like Content-Type, User-Agent, etc.
  + Body: Used in POST/PUT to send data

🔹 HTTP Response

* Server’s reply to the client.
* Format:

CODE:

HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: 123

CODE:

{

"id": 1,

"name": "John"

}

* Key Components:
  + Status Code: e.g., 200 (OK), 404 (Not Found), 500 (Error)
  + Headers: Like Content-Type, Cache-Control
  + Body: The actual response data

**2. Explain the Need and Benefits of RESTful Web Services**

🔹 What is REST?

* REST: Representational State Transfer
* An architectural style for distributed systems using HTTP.
* Each resource is accessed via a unique URL.

🔹 RESTful Web Services:

* Use standard HTTP methods: GET, POST, PUT, DELETE.
* Communicate using JSON or XML.
* Stateless interaction.

🔹 Benefits:

* Lightweight: Efficient and fast
* Scalable & Maintainable: Simple structure
* Platform Independent: HTTP support available everywhere
* Easy to Test & Debug: Tools like Postman, curl
* Separation of Client and Server: Allows parallel development

**3. Demonstrate Implementation of RESTful Web Service using GET Method**

🔹 Step 1: Controller Setup

CODE:

package com.example.demo;

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

@RestController

public class HelloController {

@GetMapping("/hello")

public String sayHello() {

return "Hello World!";

}

@GetMapping("/greet/{name}")

public String greet(@PathVariable String name) {

return "Hello, " + name;

}

@GetMapping("/students")

public String[] getStudents() {

return new String[]{"Alice", "Bob", "Charlie"};

}

}

🔹 Step 2: Run & Test

* From Browser: Visit http://localhost:8080/hello
* From Postman: Send GET to http://localhost:8080/greet/Santosh
* Auto JSON Conversion: String[] or POJOs will be returned as JSON

**4. Demonstrate End-to-End Testing of RESTful Web Service using MockMVC**

🔹 Step 1: Add Dependencies in pom.xml

CODE:

<dependency>

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

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

🔹 Step 2: Test Class

CODE:

package com.example.demo;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

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

import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest

@AutoConfigureMockMvc

public class HelloControllerTest {

@Autowired

private MockMvc mockMvc;

@Test

public void testSayHello() throws Exception {

mockMvc.perform(get("/hello"))

.andExpect(status().isOk())

.andExpect(content().string("Hello World!"));

}

@Test

public void testGreet() throws Exception {

mockMvc.perform(get("/greet/Santosh"))

.andExpect(status().isOk())

.andExpect(content().string("Hello, Santosh"));

}

@Test

public void testStudents() throws Exception {

mockMvc.perform(get("/students"))

.andExpect(status().isOk())

.andExpect(jsonPath("$[0]").value("Alice"));

}

}

🔹 Step 3: Run the Tests

* From Eclipse/IntelliJ: Right-click → Run As → JUnit Test
* From Command Line:

CODE:

mvn test