**Objectives**

* Explain in detail about HTTP Request and Response
  + HTTP Request and Response, HTTP Request Format, HTTP Response Format, Request URL, Request Method, Content-Type, User-Agent
    - Ref - https://en.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol

HTTP (Hypertext Transfer Protocol) is a protocol used for communication between a client and server. The client sends an HTTP request and the server returns an HTTP response.

**HTTP Request Format:**

An HTTP request is composed of:

* **Request Line**: Contains the HTTP method (e.g., GET, POST), the URL, and the HTTP version.
* **Headers**: Includes metadata such as Content-Type, User-Agent, etc.
* **Body**: Contains data sent to the server (mainly in POST, PUT requests).

Example:

GET /api/employees HTTP/1.1

Host: localhost:8080

User-Agent: PostmanRuntime/7.28.4

Content-Type: application/json

**HTTP Response Format:**

An HTTP response is composed of:

* **Status Line**: Contains HTTP version, status code (200 OK, 404 Not Found).
* **Headers**: Contains metadata like Content-Type, Content-Length.
* **Body**: The actual response data (often in JSON or HTML).

Example:

HTTP/1.1 200 OK

Content-Type: application/json

{

"id": 1,

"name": "John"

}

* Explain the need and benefits of RESTful Web Services
  + REST stands for REpresentational State Transfer, lightweight, maintainable, scalable, underlying protocol is HTTP; composed of resources, verbs, header, body, response status code, client-server technology
    - Ref - https://www.chakray.com/advantages-of-rest-api/

REST (Representational State Transfer) is an architectural style for building web services. RESTful services are stateless, scalable, lightweight, and rely on HTTP as the underlying protocol.

**Key Benefits:**

* **Statelessness**: No client session is stored on the server.
* **Scalability**: Easy to scale horizontally due to stateless nature.
* **Simplicity**: Easy to understand and implement using HTTP methods like GET, POST, PUT, DELETE.
* **Maintainability**: Well-organized and resource-based structure.
* **Interoperability**: Easily consumed by various clients (web, mobile, Postman).

A RESTful service uses:

* **Resources**: Represented as URIs (e.g., /employees).
* **HTTP Verbs**: GET (retrieve), POST (create), PUT (update), DELETE (remove).
* **Headers** and **Body** for input and output.
* **Status Codes** to indicate the result (200 OK, 400 Bad Request, etc.).
* Demonstrate implementation of RESTful Web Service using GET method
  + @RestController, @GetMapping, invoking get request from browser, invoking get request from postman, bean transformation to JSON, get method with parameter, return array, @PathVariable
    - Dispatcher Servlet - https://docs.spring.io/spring/docs/5.1.9.RELEASE/spring-framework-reference/web.html#mvc-servlet
    - Spring REST (Getting Started) - https://spring.io/guides/gs/rest-service/
    - Request Mapping - https://docs.spring.io/spring/docs/5.1.9.RELEASE/spring-framework-reference/web.html#mvc-ann-requestmapping

**Steps:**

1. Create a Spring Boot application with spring-boot-starter-web dependency.
2. Create a model class:

public class Employee {

private int id;

private String name;

// constructors, getters, setters

}

1. Create a REST controller:

@RestController

public class EmployeeController {

@GetMapping("/employee")

public Employee getEmployee() {

return new Employee(1, "John Doe");

}

@GetMapping("/employees")

public List<Employee> getAllEmployees() {

return Arrays.asList(

new Employee(1, "John"),

new Employee(2, "Jane")

);

}

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

public Employee getById(@PathVariable int id) {

return new Employee(id, "Name " + id);

}

}

**Testing:**

* Access via browser: http://localhost:8080/employee
* Access via Postman (GET request)
* Output will be returned in JSON format due to automatic bean-to-JSON conversion by Spring using Jackson.
* Demonstrate implementation of end to end testing of RESTful Web Service using MockMVC
  + @AutoConfigureMockMvc, MockMvc, @Test, get(), perform(), andExpect(), status().isOk(), jsonPath().exists, jsonPath().value(), status().isBadRequest(), status().reason test execution in Eclipse, test execution in command line using maven
    - Server Side Testing - https://docs.spring.io/spring/docs/5.1.9.RELEASE/spring-framework-reference/testing.html#spring-mvc-test-server

Spring Boot provides MockMvc to perform testing without starting the server.

**Steps:**

1. Add the following dependencies:
   * spring-boot-starter-test
2. Write the test:

@SpringBootTest

@AutoConfigureMockMvc

public class EmployeeControllerTest {

@Autowired

private MockMvc mockMvc;

@Test

public void testGetEmployee() throws Exception {

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").exists())

.andExpect(jsonPath("$.name").value("John Doe"));

}

@Test

public void testInvalidUrl() throws Exception {

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

.andExpect(status().isNotFound());

}

}

1. Run tests:
   * From Eclipse using JUnit test runner
   * From command line: mvn test

**Assertions Used:**

* status().isOk() – Verifies HTTP 200 status
* jsonPath().exists() – Checks if a field exists in JSON response
* jsonPath().value() – Checks the value of a field
* status().isBadRequest() or .isNotFound() – For negative test cases