**Week4\_"Hello World" RESTful Web Service**

#### Implementation Steps

#### 1. Controller Creation

A new controller class was created to handle incoming web requests.

* Class: HelloController.java
* Package: com.cognizant.spring-learn.controller
* Annotation: The class was annotated with @RestController to enable REST request handling.

**package** com.cognizant.spring\_learn.controller;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

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

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

**import** com.cognizant.spring\_learn.SpringLearnApplication;

@RestController

**public** **class** HelloController {

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(SpringLearnApplication.**class**);

@GetMapping("/hello")

**public** String sayHello() {

***LOGGER***.info("START - sayHello() method");

// The returned string will be sent as the HTTP response body.

String response = "Hello World!!";

***LOGGER***.info("END - sayHello() method");

**return** response;

}

}

#### 2. Endpoint Implementation

A method was implemented within the controller to define the specific API endpoint.

* Method Signature: public String sayHello()
* URL Mapping: The method was mapped to the URL /hello for HTTP GET requests using the @GetMapping("/hello") annotation.
* Logic: The method returns the hardcoded string "Hello World!!".
* Logging: Start and end INFO-level log statements were included as required.

#### 3. Application Configuration

The application's port was configured to match the requirement.

* File: application.properties
* Property: server.port=8083

spring.application.name=spring-learn

server.port=8083

### Testing and Verification

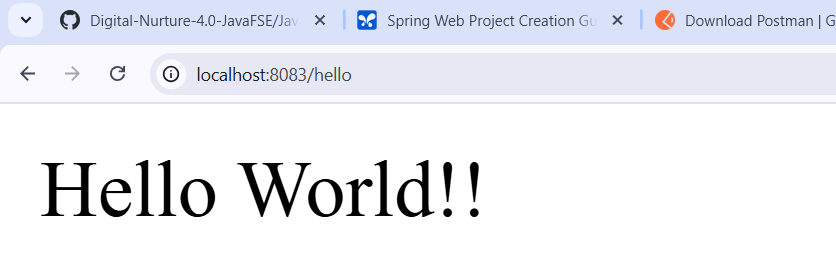
#### 1. Execution

The spring-learn project was launched as a Spring Boot application. The embedded Tomcat server started successfully on the configured port 8083.

#### 2. Test Environment 1: Web Browser (Chrome)

* Request: Navigated to http://localhost:8083/hello.
* Result: The browser correctly displayed the response body: Hello World!!.
* Header Analysis: Using Chrome Developer Tools (F12), the "Network" tab showed the response headers, including the Content-Type as text/plain.

**Output**



#### 3. Test Environment 2: Postman

* Request: A new GET request was sent to http://localhost:8083/hello.
* Result: The "Body" of the response correctly contained the text Hello World!!.
* Header Analysis: The "Headers" tab in the Postman response view confirmed the HTTP headers (e.g., Content-Type, Content-Length, Date) were received as expected.

**Output**

