**Spring REST using Spring Boot**

**Handson 1 – Create a Spring Web Project using Maven**

**1. Downloaded spring starter JAR zip and extracted in Eclipse Workspace.**

**2. Walked through the aspects related to project**

**src/main/java**

* Contains all Java source code.
* Includes com.cognizant.spring\_learn package and SpringLearnApplication.java.

**src/main/resources**

* For application config files.
* Example: application.properties, .yml, XML configs, static files, templates.

**src/test/java**

* Contains unit and integration test classes.
* Mirrors the structure of src/main/java.

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

System.out.println("Application Started successfully");

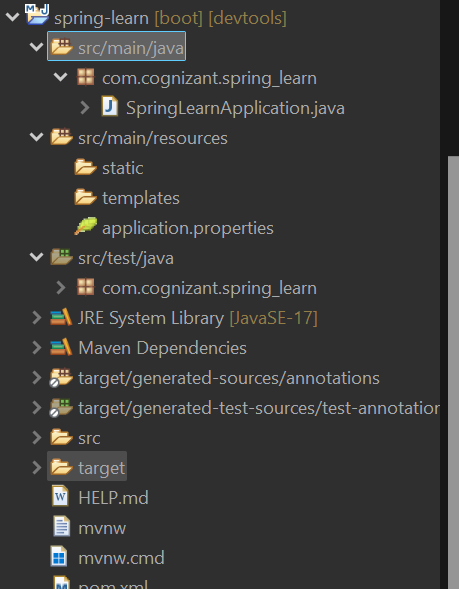
}

}

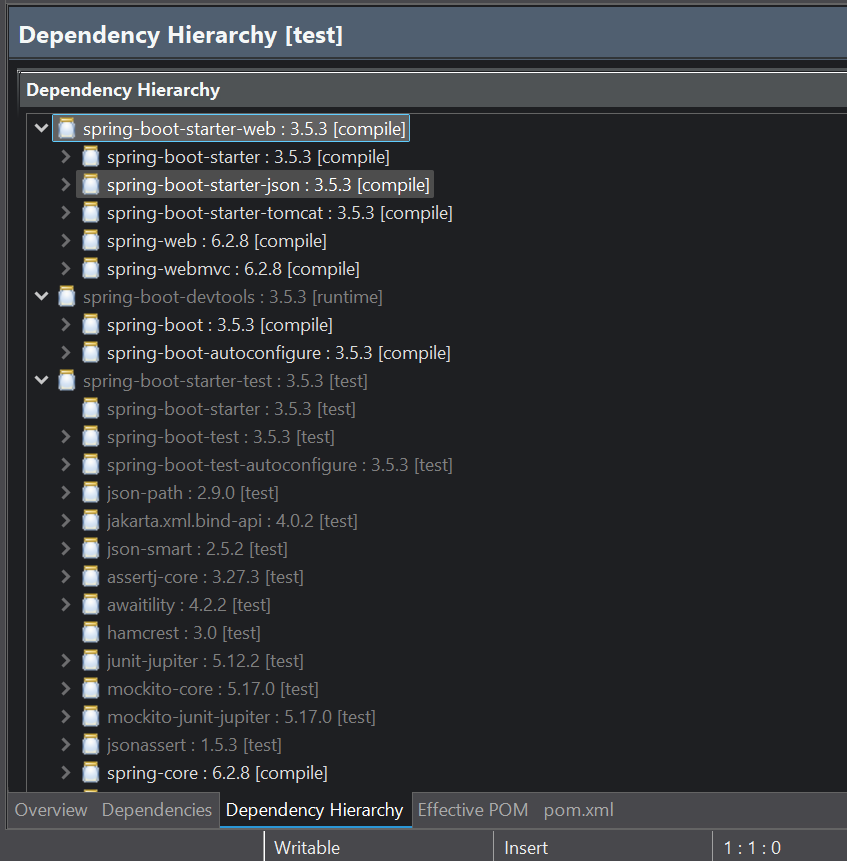
**@SpringBootApplication Annotation**

* + @Configuration - Configures Spring beans.
  + @EnableAutoConfiguration - Enables auto-config.
  + @ComponentScan - Scans com.cognizant.springlearn package and sub-packages for components.

**Folder Structure:**

****

**Dependency Hierarchy Tree:**



**Hands on 2 - Spring Core Load SimpleDateFormat from Spring Configuration XML**

**1. Created a spring configuration file date-format.xml and Defined Bean tags**

**Code:**

<?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*

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

<bean id=*"dateFormat"* class=*"java.text.SimpleDateFormat"*>

<constructor-arg value=*"dd/MM/yyyy"*/>

<!-- collaborators and configuration for this bean go here -->

</bean>

<!-- more bean definitions go here -->

</beans>

**2. Created a new method displayDate() in SpringLearnApplication.java**

**Code:**

package com.cognizant.spring\_learn;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

*@SpringBootApplication*

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.*run*(SpringLearnApplication.class, args);

System.*out*.println("Application Started successfully");

ApplicationContext context =new ClassPathXmlApplicationContext("date-format.xml");

SimpleDateFormat format = context.getBean("dateFormat",SimpleDateFormat.class);

Date date;

try {

date = format.parse("18/04/2005");

System.*out*.print(date);

} catch (ParseException e) {

e.printStackTrace();

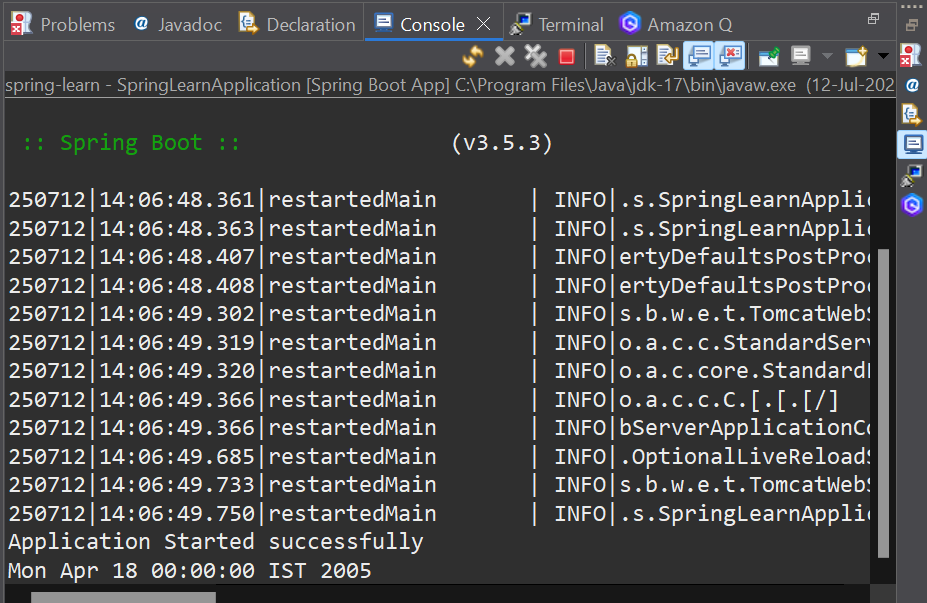
}

} }

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

****

**Hands on 3 - Spring Core - Incorporate Logging**

1. **Configured Application properties and enabled debugging**

spring.application.name=spring-learn

server.port = 3000

logging.level.org.springframework=info

logging.level.com.cognizant.spring\_learn=debug

logging.pattern.console=%d{yyMMdd}|%d{HH:mm:ss.SSS}|%-20.20thread|%5p|%-25.25logger{25}|%25M|%m%n

1. **SpringApplication.java Code:**

**Code:**

package com.cognizant.spring\_learn;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

*@SpringBootApplication*

public class SpringLearnApplication {

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

private static ApplicationContext *context*;

public static void main(String[] args)

{

SpringApplication.*run*(SpringLearnApplication.class, args);

*LOGGER*.info("Start");

*displayDate*();

*LOGGER*.info("END");

}

public static void displayDate() {

*LOGGER*.info("Start");

ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");

SimpleDateFormat format = context.getBean("dateFormat",SimpleDateFormat.class);

Date date;

try {

date = format.parse("18/04/2005");

*LOGGER*.debug("Date: {}",date.toString());

} catch (ParseException e) {

e.printStackTrace();

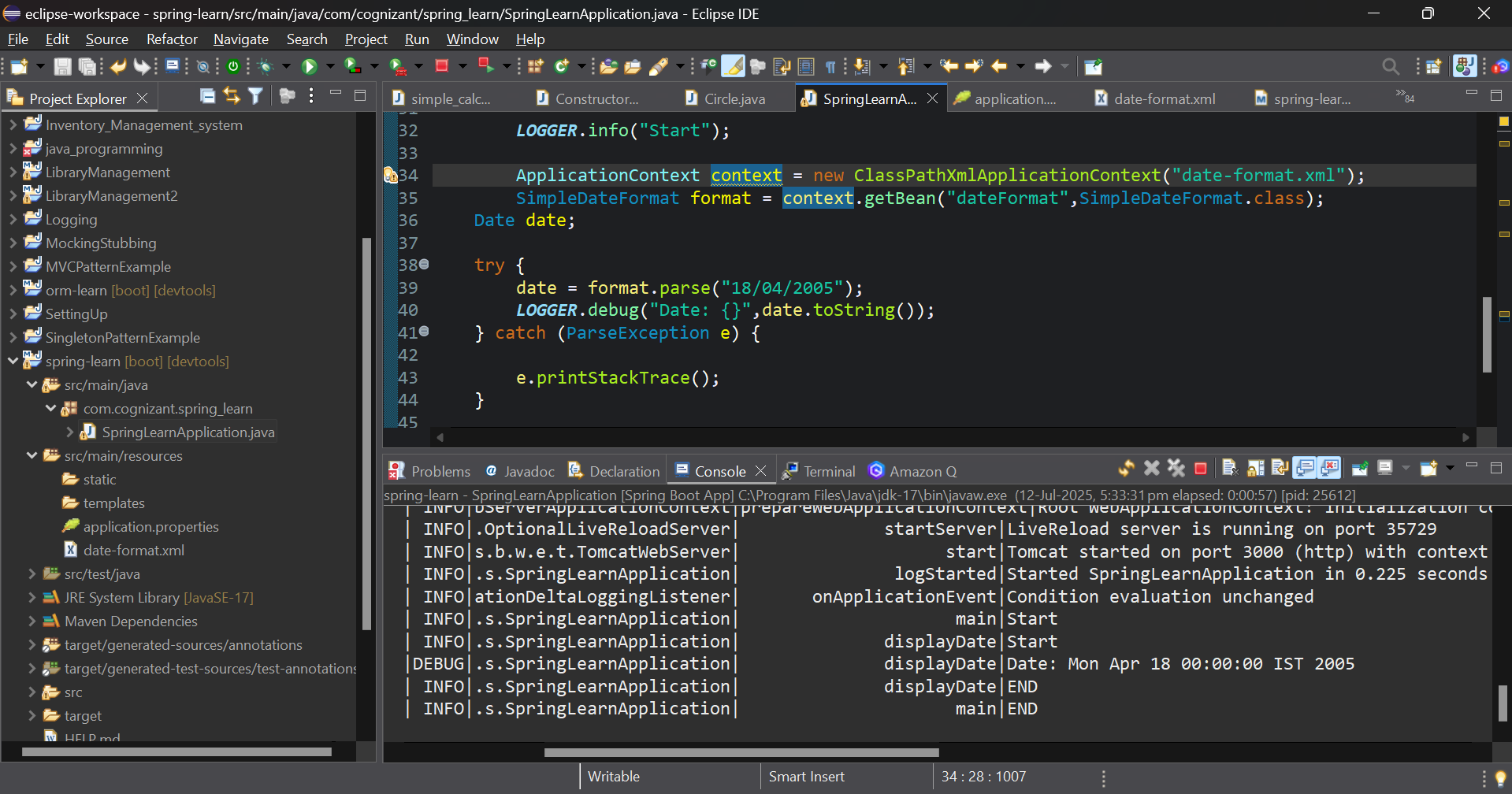
}

*LOGGER*.info("END");

}

}

**Output:**



A screen shot of a computer program

AI-generated content may be incorrect.

**Hands on 4 - Spring Core – Load Country from Spring Configuration XML**

**1. Created country.xml and Created Beans of Country class.**

**Code:**

<?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*

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

<bean id=*"country1"* class=*"com.cognizant.spring\_learn.Country"*>

<property name=*"code"* value=*"IN"* />

<property name=*"name"* value=*"India"* />

</bean>

<bean id=*"country2"* class=*"com.cognizant.spring\_learn.Country"*>

<property name=*"code"* value=*"US"* />

<property name=*"name"* value=*"United States"* />

</bean>

</beans>

**2. Created Country class with name and code Aspect, Getter and Setter methods with debug logs.**

**Code:**

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Country{

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

private String code;

private String name;

public Country()

{

*LOGGER*.debug("Inside Country Constrcutor");

}

public String getCode()

{

*LOGGER*.debug("Getting the Country Code:{}",code);

return code;

}

public void setName(String name)

{ *LOGGER*.debug("Setting Name of the country as: {}", name);

this.name=name;

}

public void setCode(String code)

{

*LOGGER*.debug("Setting code of the country as: {}", code);

this.code=code;

}

public String getName()

{

*LOGGER*.debug("The Country Name is: {}",name);

return name;

}

*@Override*

public String toString()

{

return "Country[code = " + code + "," +" name = "+name+"]";

}

}

**3. Create SpringLearnCountryApplication Class as SpringBootApplication File**

**Code:**

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

*@SpringBootApplication*

public class SpringLearnCountryApplication {

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

public static void main(String[] args) {

SpringApplication.*run*(SpringLearnApplication.class, args);

*LOGGER*.info("START");

*displayCountry*();

*LOGGER*.info("END");

}

public static void displayCountry()

{

*LOGGER*.info("Start");

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

Country country1 = context.getBean("country1", Country.class);

Country country2 = context.getBean("country2",Country.class);

*LOGGER*.debug("Country : {}", country1);

*LOGGER*.debug("Country : {}",country2);

*LOGGER*.info("END");

}

}

**4. SME Explanation Section:**

1. **bean tag:**

<bean id="country" class="com.cognizant.spring\_learn.Country">

* + Defines a Spring-managed object.
  + id – unique name to reference the bean ("country").

class – fully qualified class name.

1. **property tag:**

<property name="code" value="IN"/>

* + Used to set values into the bean fields via setter methods.
  + name – must match the Java setter name (setCode).
  + value – the actual value assigned ("IN").

1. **ApplicationContext and ClassPathXmlApplicationContext:**

* ApplicationContext is the Spring container that loads beans.
* ClassPathXmlApplicationContext loads an XML config file from the classpath.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

1. **What happens when context.getBean(...) is called?**

Country country = context.getBean("country", Country.class);

* Spring instantiates the bean (Country object).
* It calls the constructor.
* Then it calls the setters with values from <property>.
* Returns the fully initialized object.

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

**Handson - Hello World RESTful Web Service**

1. **Created HelloController in com.cognizant.spring\_learn.controller package.**

**Code:**

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;

*@RestController*

public class HelloController {

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

*@GetMapping*("/hello")

public String sayHello() {

*LOGGER*.info("START");

String message = "Hello World!!";

*LOGGER*.info("END");

return message;

}

}

1. **Set the server port to 8083 in application.properties.**
2. **SpringLearnApplication class is configured**

**Code:**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.*run*(SpringLearnApplication.class, args);

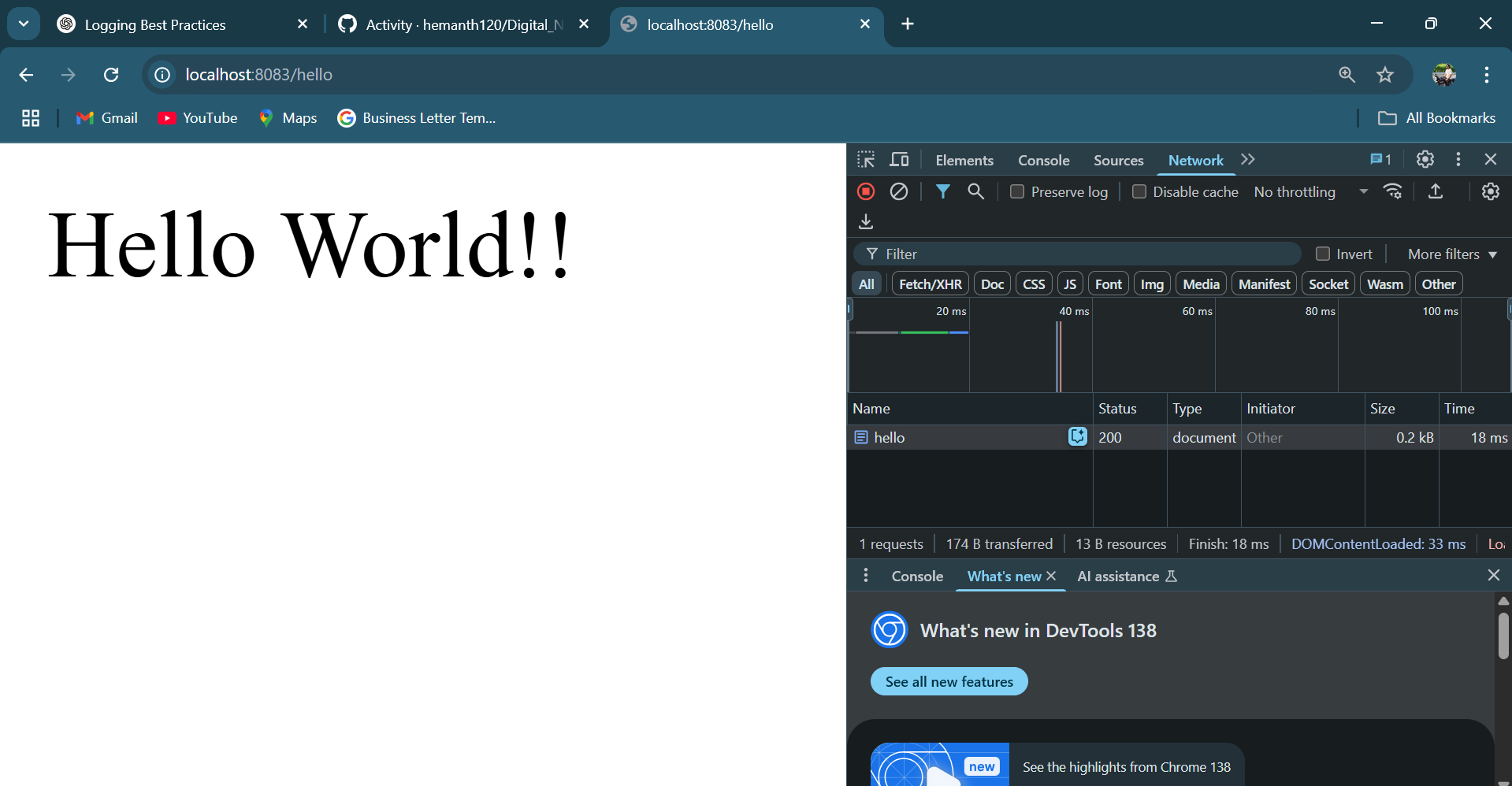
}

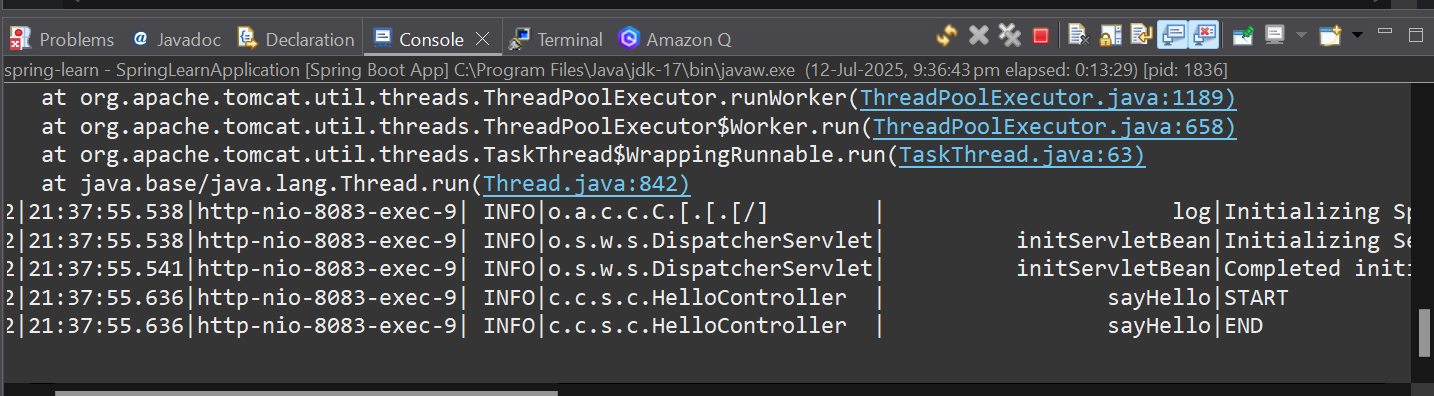
}

1. **Ran the SpringLearnApplication and tried to access the**

[**http://localhost:8083/hello**](http://localhost:8083/hello) **URL**

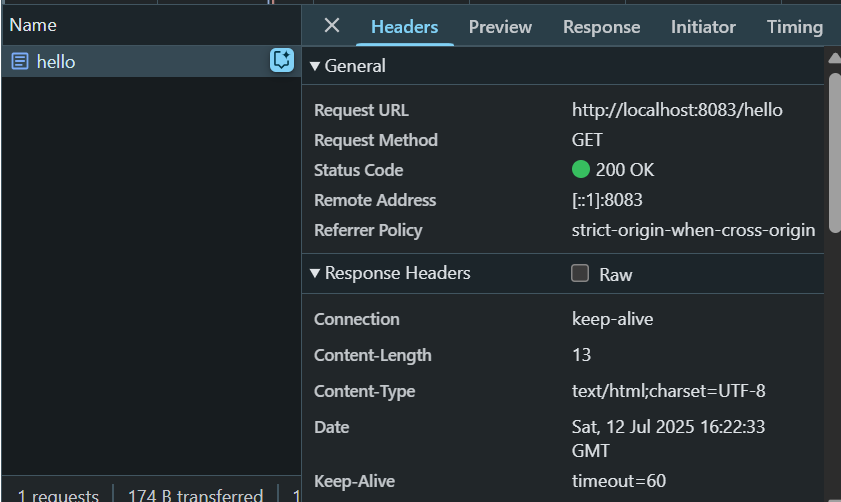
**Output:**



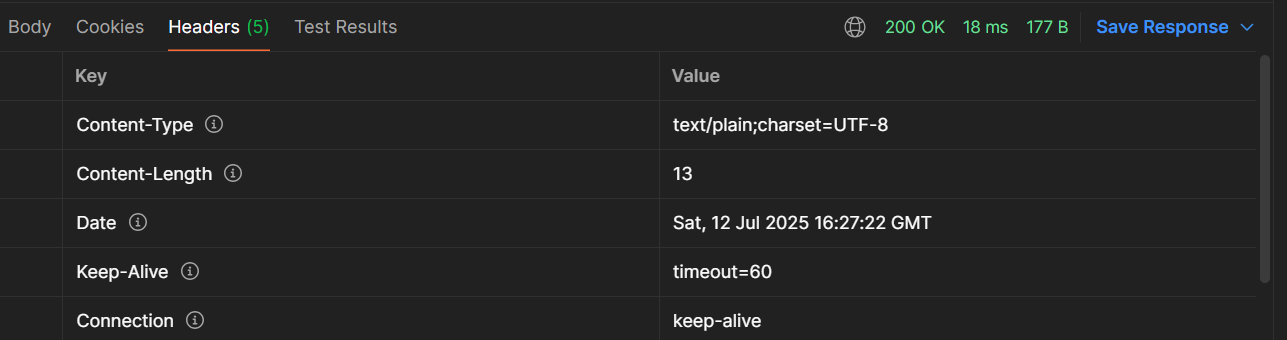


1. **SME Explanation**

* **In network tab of developer tools, the HTTP header details received**



* **In postman, the HTTP header details received in the Header tab.**



**Handson - Country Web Service**

**1. Created a CountryContoller class in the controller package.**

**Code:**

package com.cognizant.spring\_learn.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

import com.cognizant.spring\_learn.Country;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

*@RestController*

public class CountryController{

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

*@RequestMapping*("/country")

public Country getIndia()

{ *LOGGER*.info("Start");

ApplicationContext context=new ClassPathXmlApplicationContext("country.xml");

Country country = context.getBean("country1",Country.class);

*LOGGER*.info("END");

return country;

}

}

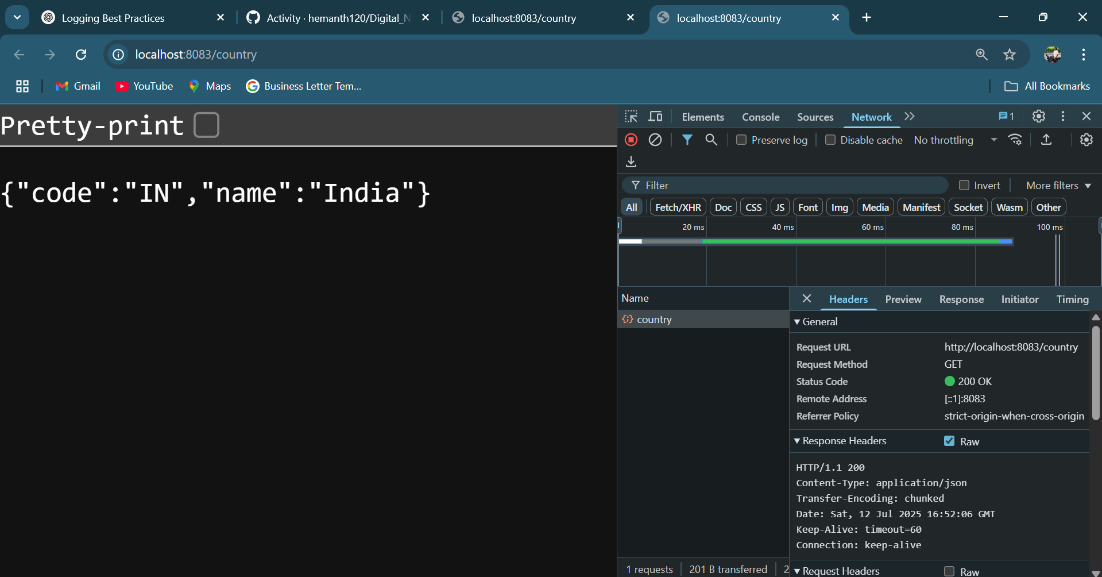
**2. Used already existing Country class and beans.**

**3. Set the server port to 8083 since the previous handson**

**4. Ran the SpringLearnAppliation.java and tried to access**

<http://localhost:8083/hello> url

**Output:**



**5. SME Explanation:**

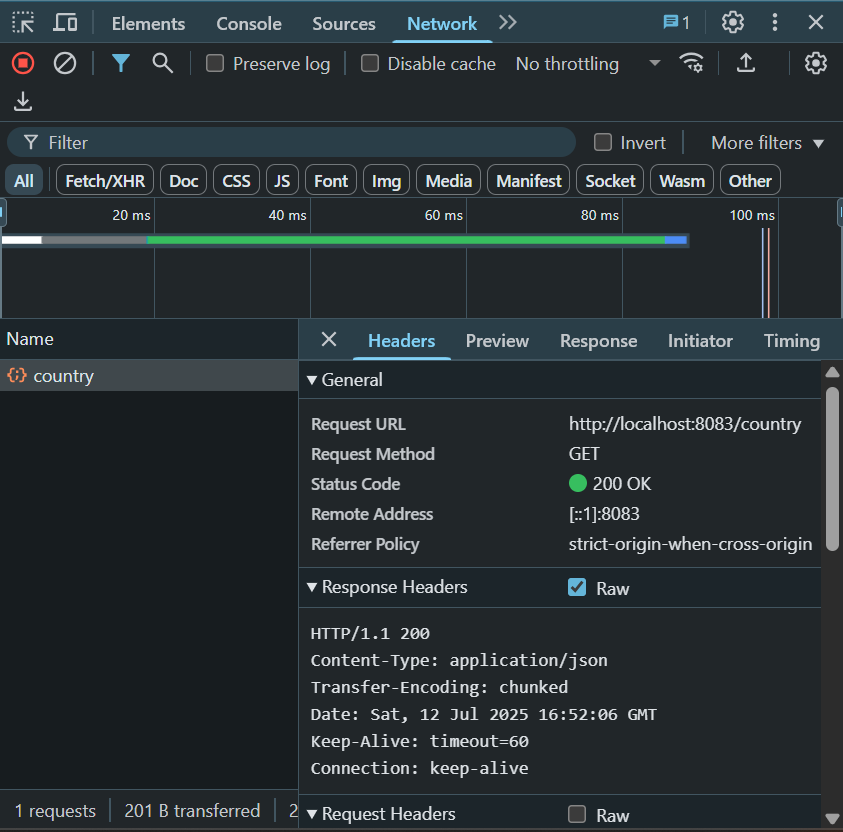
1. **What happens in the controller method?**

* The method getCountryIndia() is called when /country is requested.
* It loads country.xml using ClassPathXmlApplicationContext.
* Retrieves the country bean.
* Returns it — Spring automatically converts the returned Country object into JSON.

1. **How is the bean converted to JSON?**

* Spring Boot includes Jackson (a JSON mapper) by default.
* The @RestController and @RequestMapping return objects.

1. **In network tab of developer tools, the HTTP header details received**



1. **In postman, the HTTP header details received on Header tab**

**A screenshot of a computer

AI-generated content may be incorrect.**