WEEK - 4 Hands On

*Hands on 1*

Create a Spring Web Project using Maven   
  
Follow steps below to create a project:

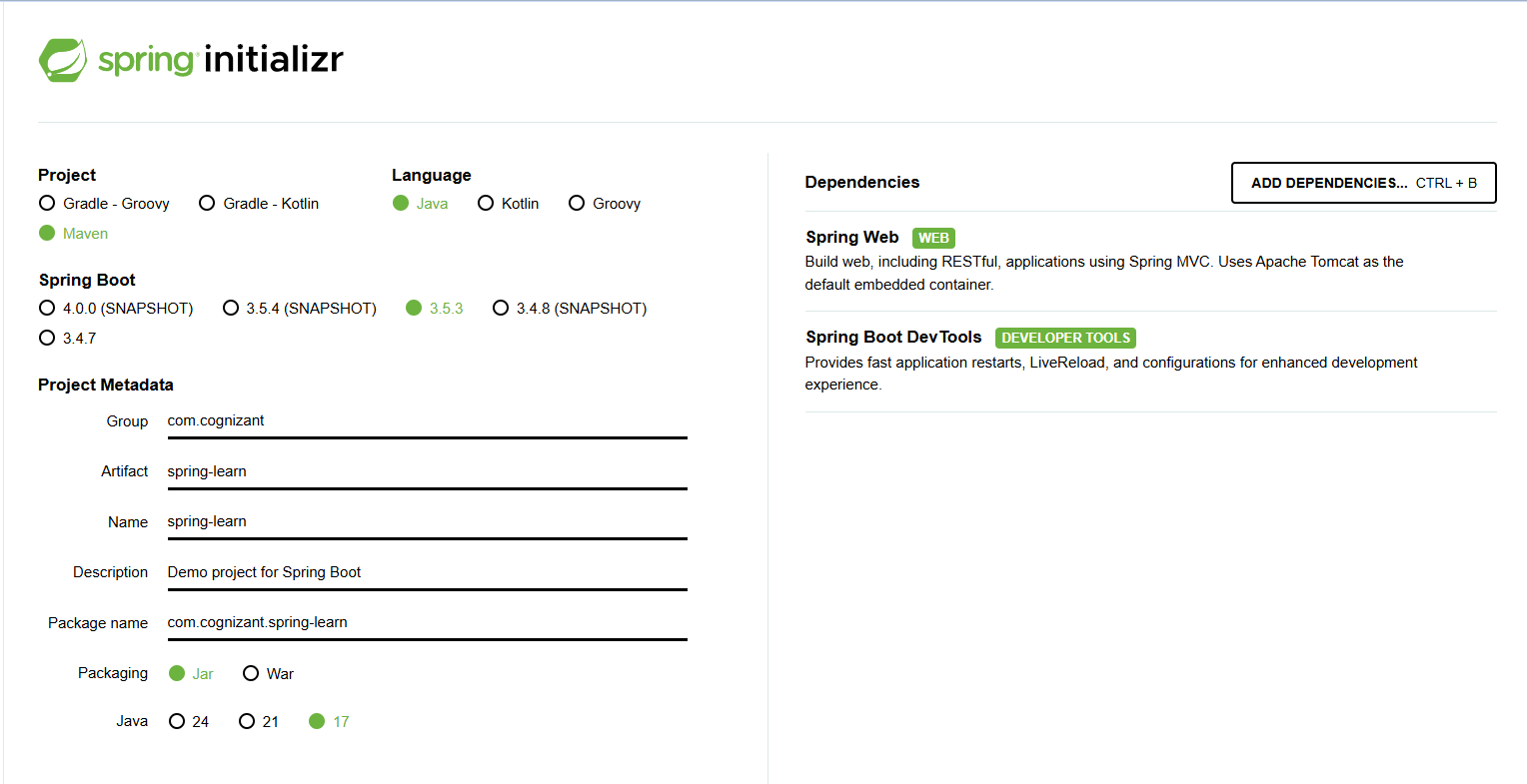
1. Go to <https://start.spring.io/>
2. Change Group as “com.cognizant”
3. Change Artifact Id as “spring-learn”
4. Select Spring Boot DevTools and Spring Web
5. Create and download the project as zip
6. Extract the zip in root folder to Eclipse Workspace
7. Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
8. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
9. Include logs to verify if main() method of SpringLearnApplication.
10. Run the SpringLearnApplication class.
11. SME to walk through the following aspects related to the project created:
12. src/main/java - Folder with application code
13. src/main/resources - Folder for application configuration
14. src/test/java - Folder with code for testing the application
15. SpringLearnApplication.java - Walkthrough the main() method.
16. Purpose of @SpringBootApplication annotation
17. pom.xml
    1. Walkthrough all the configuration defined in XML file
    2. Open 'Dependency Hierarchy' and show the dependency tree.

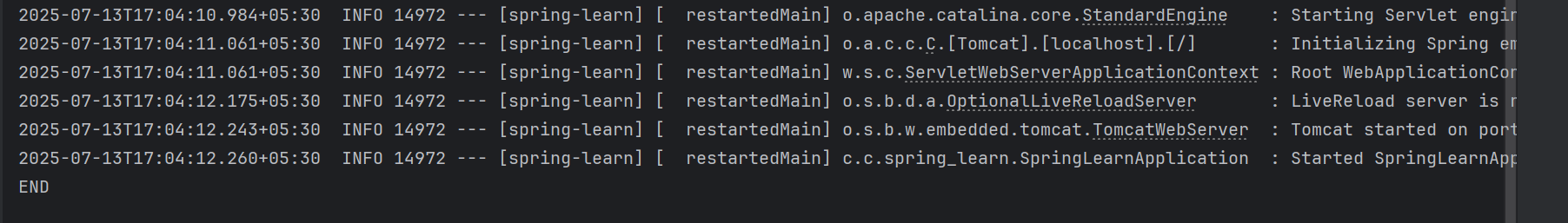
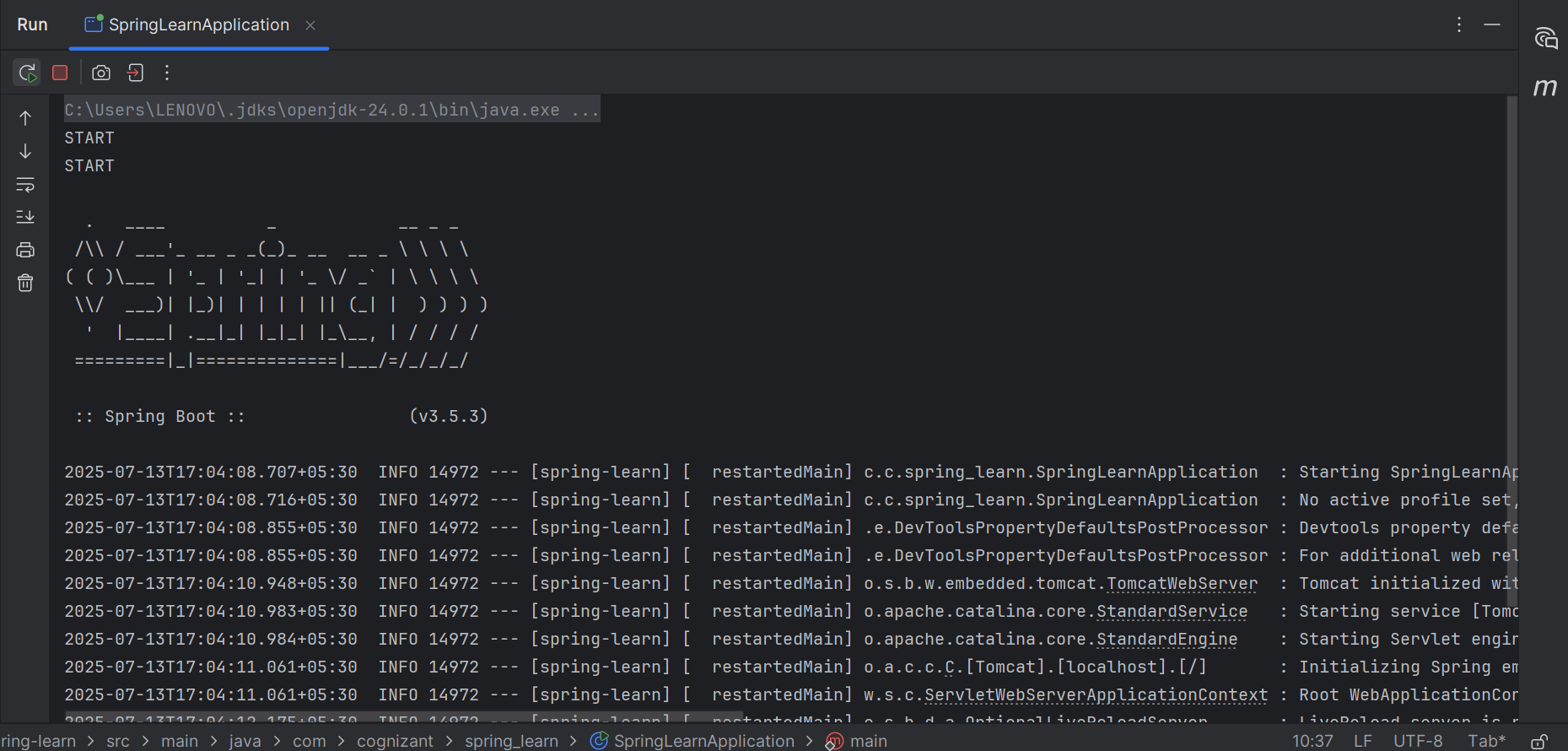
***Solution:***

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) {  
 System.*out*.println("START");  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 System.*out*.println("END");  
 }  
}

Output:





*Hands on 2*

Spring Core – Load SimpleDateFormat from Spring Configuration XML   
  
SimpleDateFormat with the pattern ‘dd/MM/yyyy’ is created in multiple places of an application. To avoid creation of SimpleDateFormat in multiple places, define a bean in Spring XML Configuration file and retrieve the date.  
  
Follow steps below to implement:

* Create spring configuration file date-format.xml in src/main/resources folder of 'spring-learn' project
* Open https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-factory-metadata
* Copy the XML defined in the section of previous step URL and paste it into date-format.xml
* Define bean tag in the XML with for date format. Refer code below.

<?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" />

</bean>

</beans>

* Create new method displayDate() in SpringLearnApplication.java
* In displayDate() method create the ApplicationContext. Refer code below:

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

* Get the dateFormat using getBean() method. Refer code below.

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

* Using the format variable try to parse string '31/12/2018' to Date class and display the result using System.out.println.
* Run the application as 'Java Application' and check the result in console log output.

**Troubleshooting Tips**   
  
If the tomcat port has a conflict and the server is not starting include the below property in application.properties file in src/main/resources folder.

*Solution:*

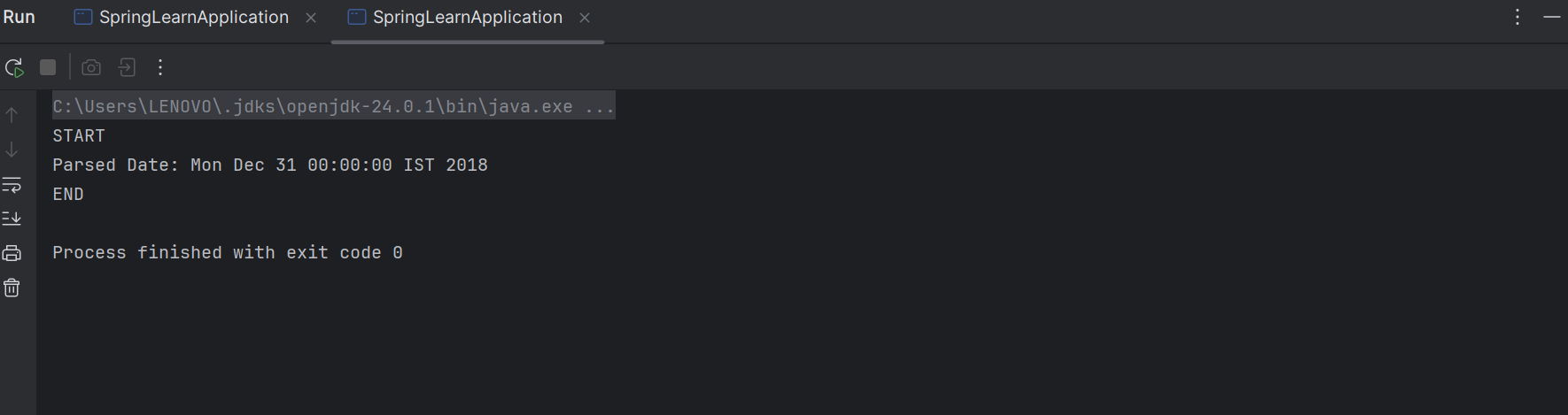
Date-format.xml

<?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" />  
 </bean>  
  
</beans>

SpringLearnApplication.java

package com.cognizant.spring\_learn;  
  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class SpringLearnApplication {  
  
 public static void main(String[] args) throws ParseException {  
 System.*out*.println("START");  
 *displayDate*();  
 System.*out*.println("END");  
 }  
  
 public static void displayDate() throws ParseException {  
 // Load Spring XML config  
 ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");  
  
 // Get the SimpleDateFormat bean  
 SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);  
  
 // Use it to parse a date string  
 String dateString = "31/12/2018";  
 Date date = format.parse(dateString);  
  
 // Print the parsed date  
 System.*out*.println("Parsed Date: " + date);  
 }  
}

Output:



*Hands On 3*

Hello World RESTful Web Service   
  
Write a REST service in the spring learn application created earlier, that returns the text "Hello World!!" using Spring Web Framework. Refer details below:  
  
**Method:** GET  
**URL:** /hello  
**Controller:** com.cognizant.spring-learn.controller.HelloController  
**Method Signature:** public String sayHello()  
**Method Implementation:** return hard coded string "Hello World!!"  
**Sample Request**: http://localhost:8083/hello  
**Sample Response:** Hello World!!   
  
**IMPORTANT NOTE**: Don't forget to include start and end log in the sayHello() method.  
  
Try the URL http://localhost:8083/hello in both chrome browser and postman.  
  
SME to explain the following aspects:

* In network tab of developer tools show the HTTP header details received
* In postman click on "Headers" tab to view the HTTP header details received

*Solutions*

Application.properties

spring.application.name=spring-learn  
server.port=8083

HelloController.java

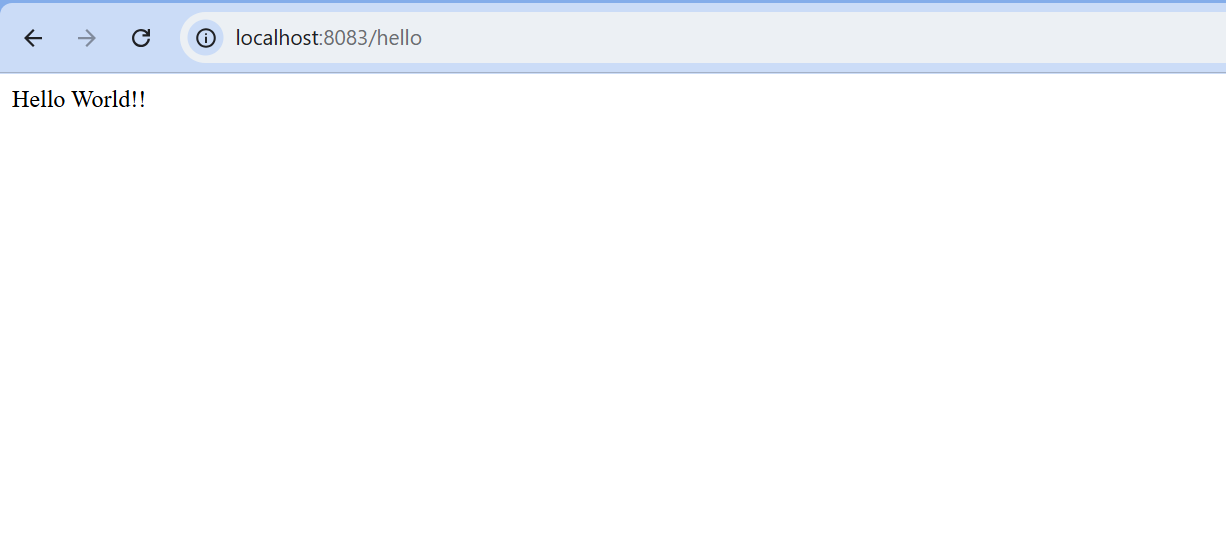
package com.cognizant.spring\_learn.controller;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class HelloController {  
  
 @GetMapping("/hello")  
 public String sayHello() {  
 System.*out*.println("START");  
 String message = "Hello World!!";  
 System.*out*.println("END");  
 return message;  
 }  
}

SpringLearnApplication.java

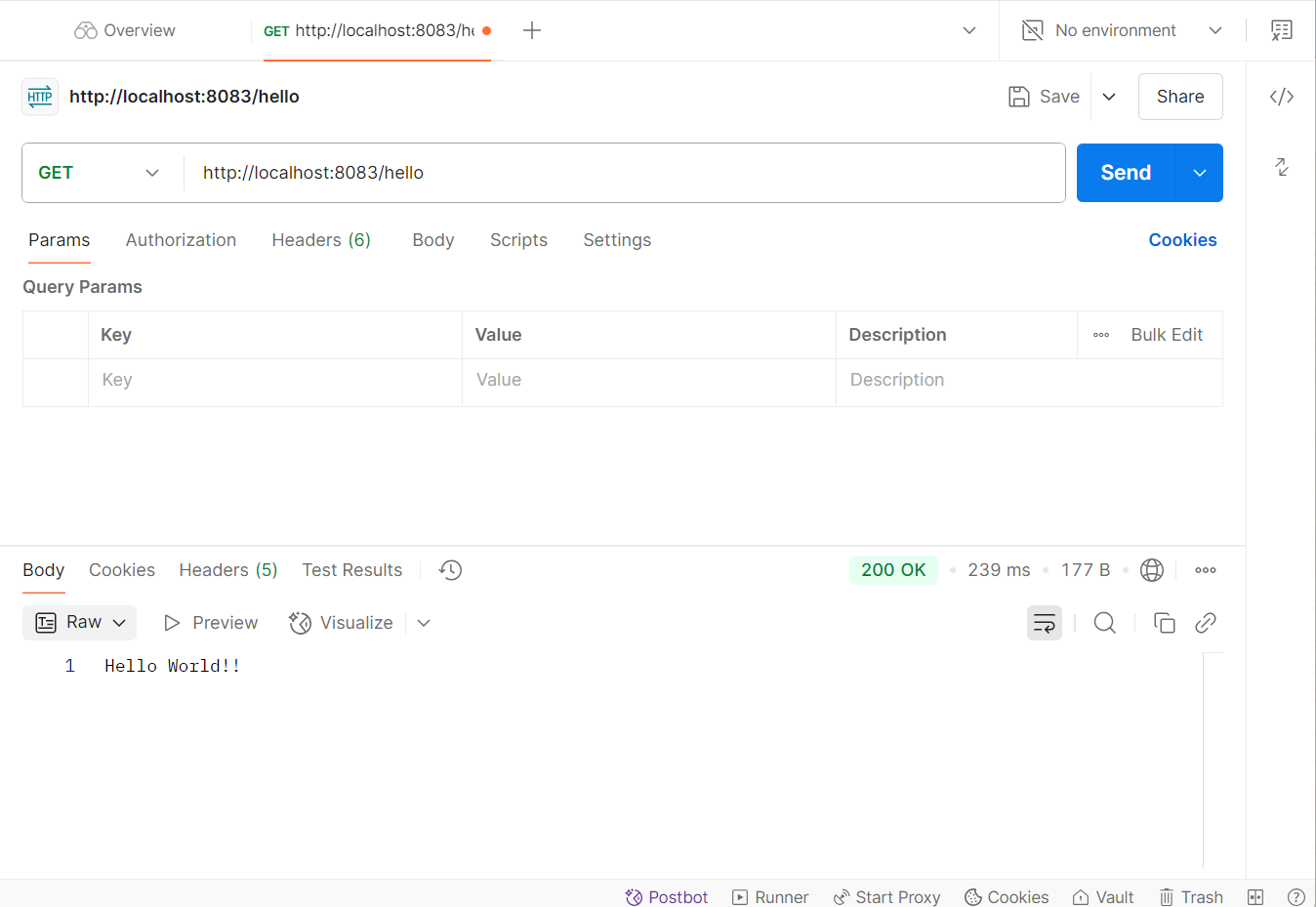
package com.cognizant.spring\_learn;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class SpringLearnApplication {  
  
 public static void main(String[] args) {  
 System.*out*.println("START");  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 System.*out*.println("END");  
 }  
}

Output:

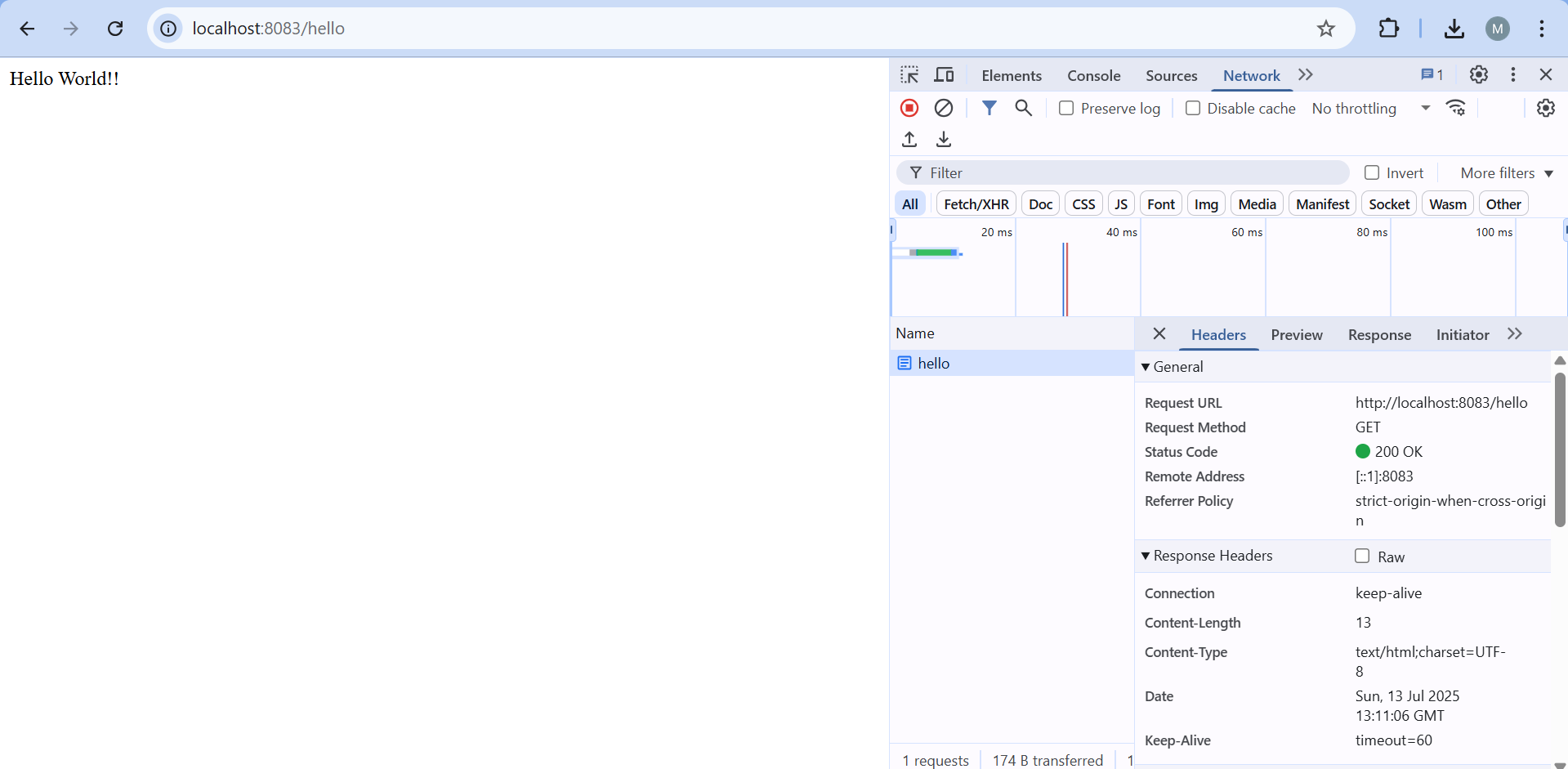
Testing the API in chrome



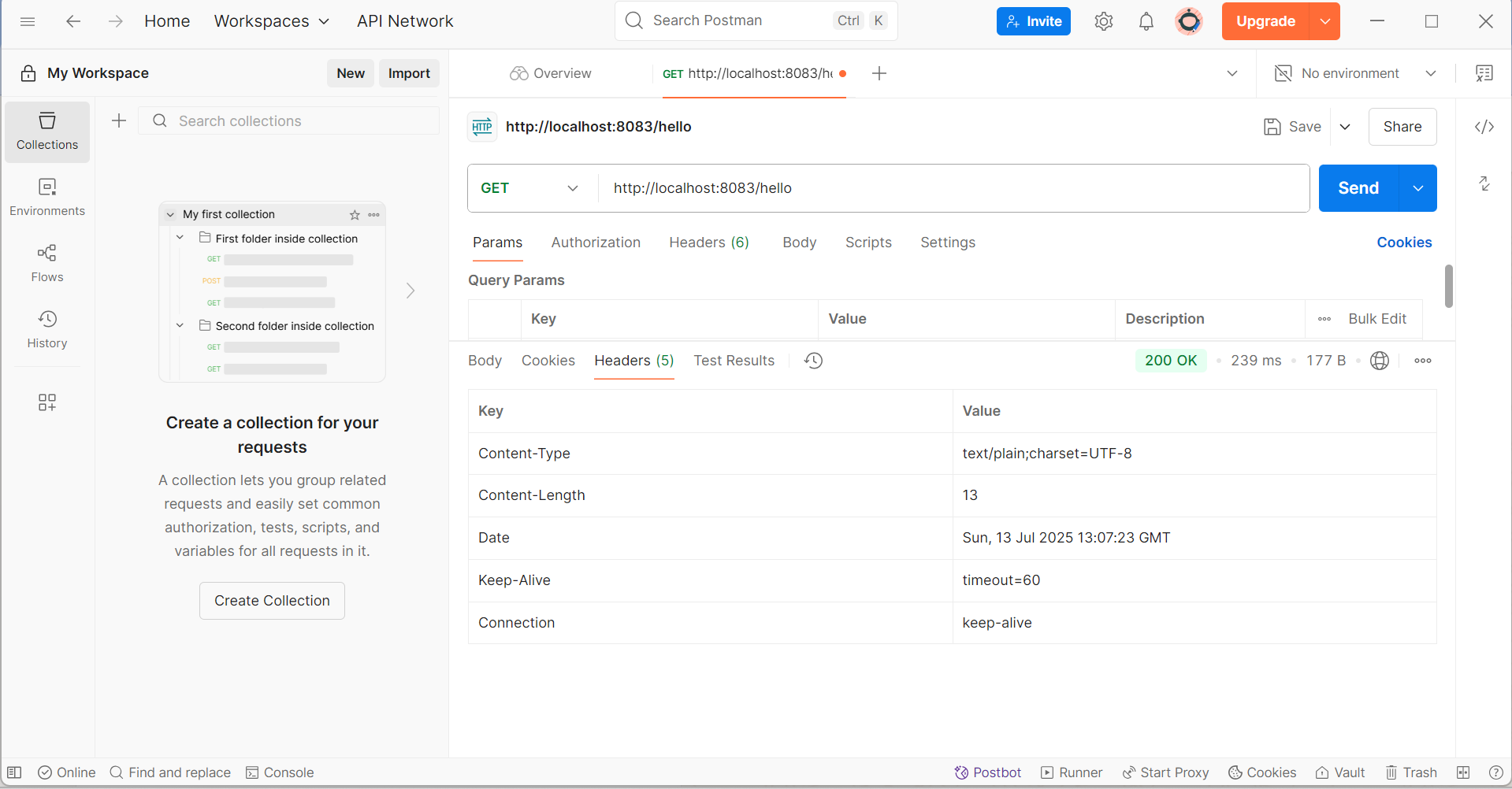
Testing the API in Postman



Chrome: View HTTP Headers



Postman: View HTTP Headers



*Hands On 4*

REST - Country Web Service   
  
Write a REST service that returns India country details in the earlier created spring learn application.  
  
**URL**: /country  
**Controller**: com.cognizant.spring-learn.controller.CountryController  
**Method Annotation**: @RequestMapping  
**Method Name**: getCountryIndia()  
**Method Implementation**: Load India bean from spring xml configuration and return  
**Sample Request**: http://localhost:8083/country  
**Sample Response**:

{

  "code": "IN",

  "name": "India"

}

SME to explain the following aspects:

* What happens in the controller method?
* How the bean is converted into JSON reponse?
* In network tab of developer tools show the HTTP header details received
* In postman click on "Headers" tab to view the HTTP header details received

*Solution*

Country.java

package com.cognizant.spring\_learn.model;  
  
public class Country {  
 private String code;  
 private String name;  
  
 public Country() {}  
  
 public Country(String code, String name) {  
 this.code = code;  
 this.name = name;  
 }  
  
 public String getCode() {  
 return code;  
 }  
  
 public void setCode(String code) {  
 this.code = code;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

country.xml

<?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="in" class="com.cognizant.spring\_learn.model.Country">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
 </bean>  
  
</beans>

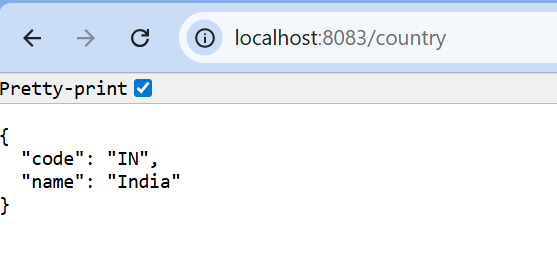
CountryController.java

package com.cognizant.spring\_learn.controller;  
  
import com.cognizant.spring\_learn.model.Country;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class CountryController {  
  
 @RequestMapping("/country")  
 public Country getCountryIndia() {  
 System.*out*.println("START");  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 Country country = context.getBean("in", Country.class);  
 System.*out*.println("END");  
 return country;  
 }  
}

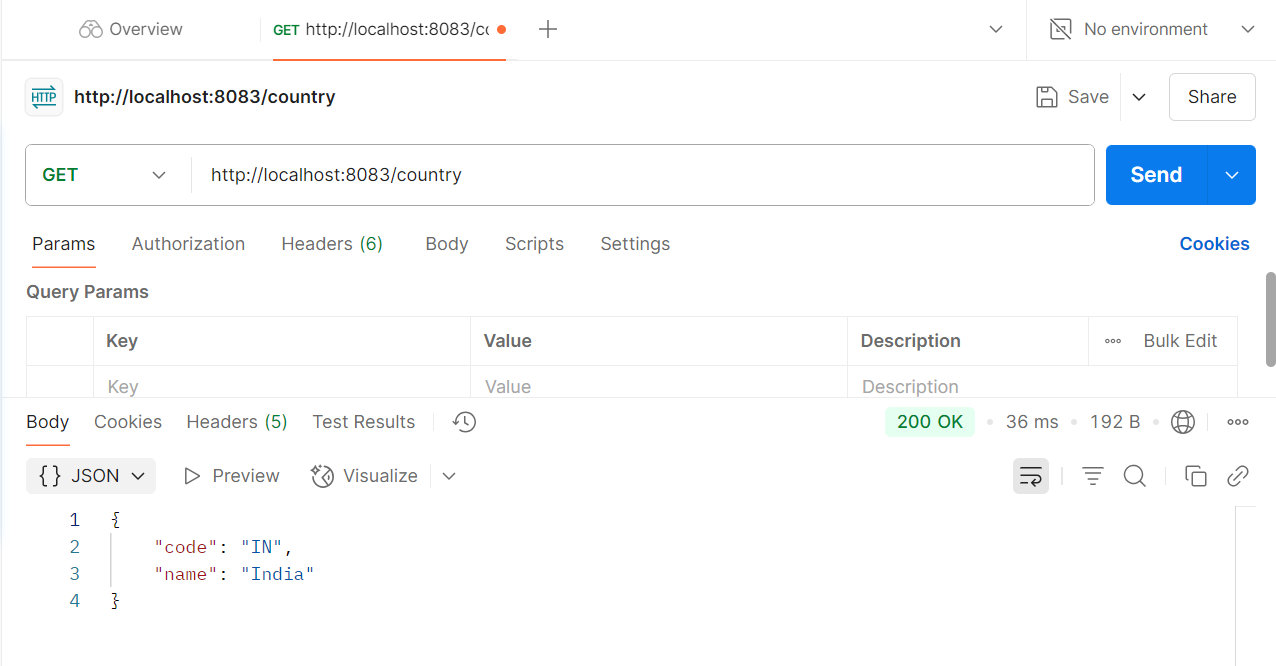
SpringLearnApplication.java

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) {  
 System.*out*.println("START");  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 System.*out*.println("END");  
 }  
}

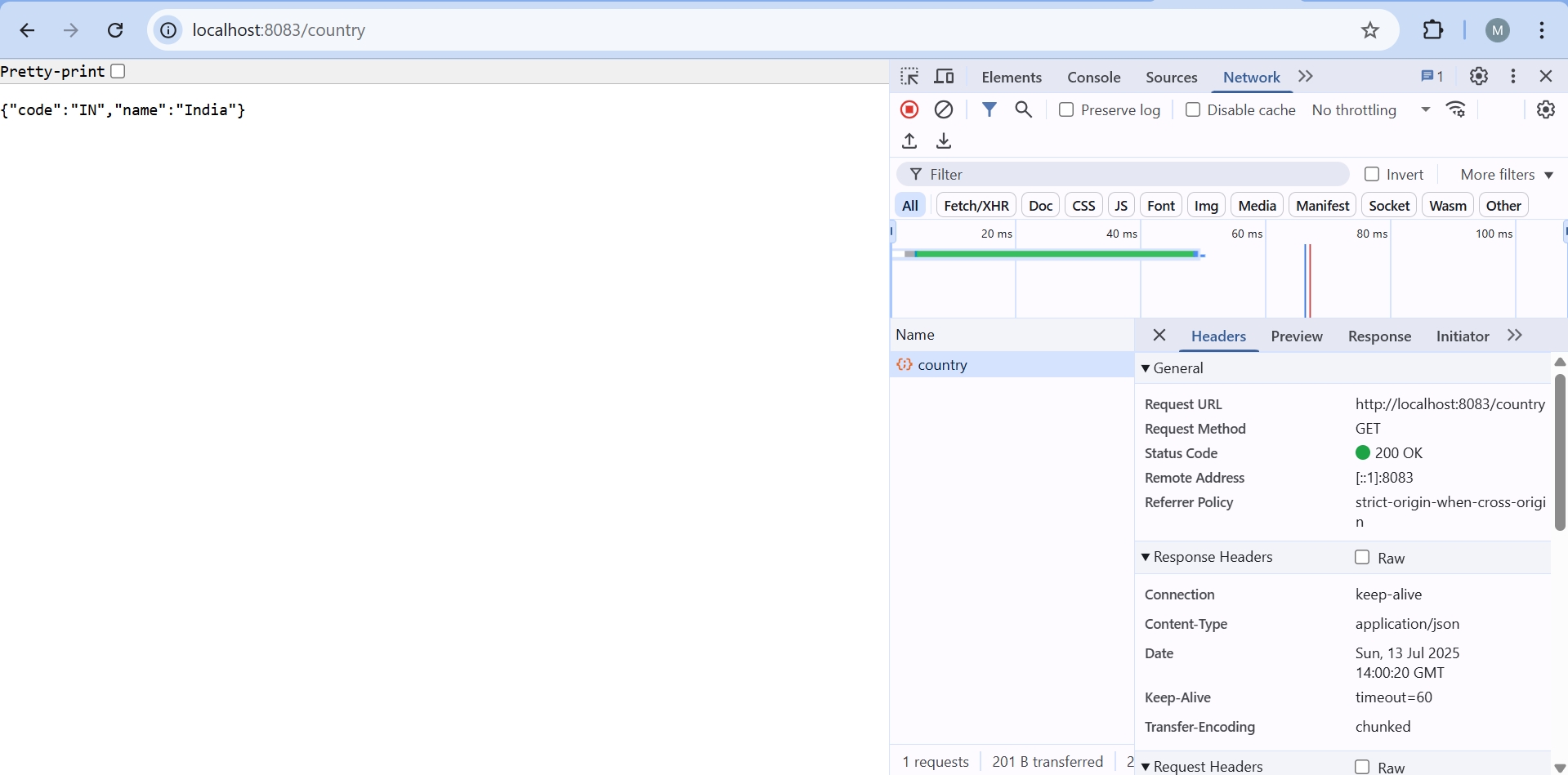
Testing the API in chrome



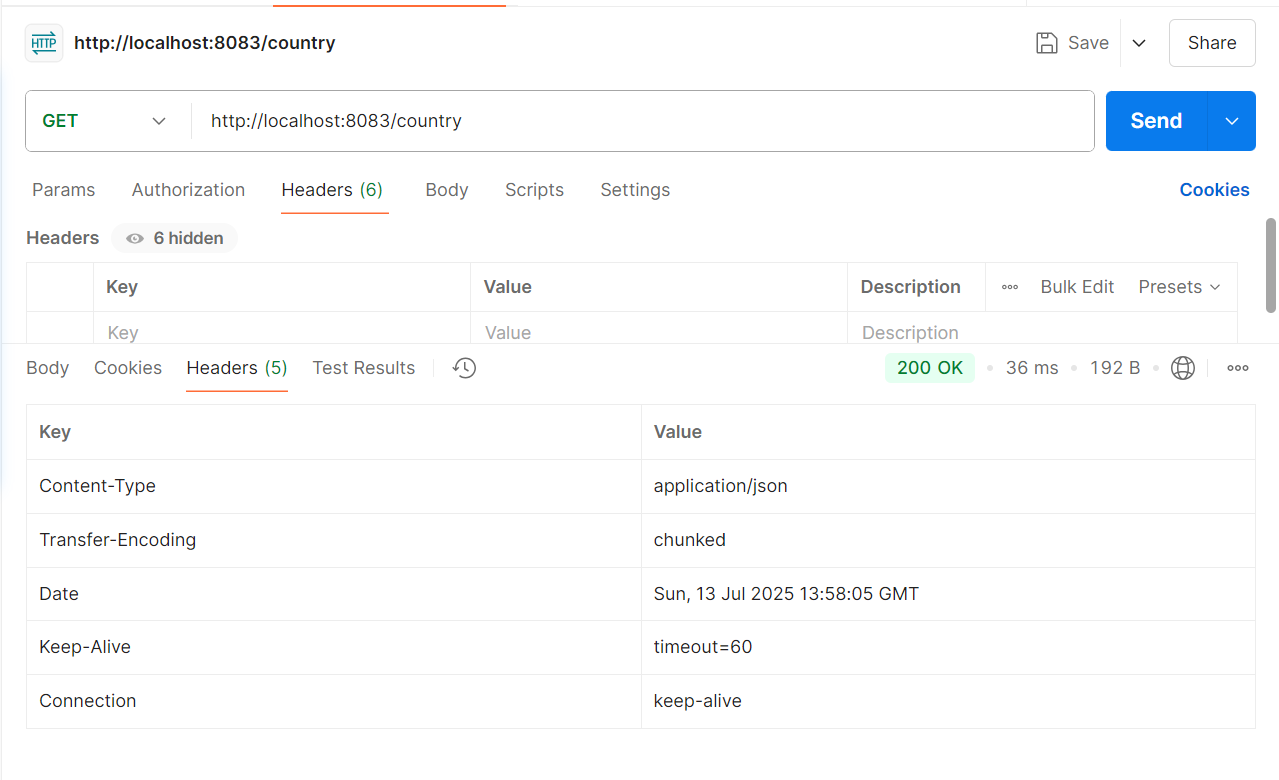
Testing the API in Postman



Chrome: View HTTP Headers



Postman: View HTTP Headers



*Hands On 5*

REST - Get country based on country code   
  
Write a REST service that returns a specific country based on country code. The country code should be case insensitive.  
  
**Controller**: com.cognizant.spring-learn.controller.CountryController  
**Method Annotation:** @GetMapping("/countries/{code}")  
**Method Name**: getCountry(String code)  
**Method Implemetation**: Invoke countryService.getCountry(code)   
**Service Method:**com.cognizant.spring-learn.service.CountryService.getCountry(String code)  
  
**Service Method Implementation**:

* Get the country code using @PathVariable
* Get country list from country.xml
* Iterate through the country list
* Make a case insensitive matching of country code and return the country.
* Lambda expression can also be used instead of iterating the country list

**Sample Request**: http://localhost:8083/country/in  
  
**Sample Response**:

{

  "code": "IN",

  "name": "India"

}

*Solution*

Country.xml

<?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="countryList" class="java.util.ArrayList">  
 <constructor-arg>  
 <list>  
 <bean class="com.cognizant.spring\_learn.model.Country">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
 </bean>  
 <bean class="com.cognizant.spring\_learn.model.Country">  
 <property name="code" value="US"/>  
 <property name="name" value="United States"/>  
 </bean>  
 <bean class="com.cognizant.spring\_learn.model.Country">  
 <property name="code" value="CN"/>  
 <property name="name" value="China"/>  
 </bean>  
 </list>  
 </constructor-arg>  
 </bean>  
  
</beans>

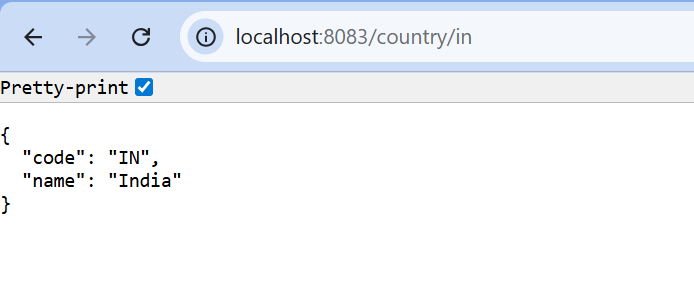
CountryService.java

package com.cognizant.spring\_learn.service;  
  
import com.cognizant.spring\_learn.model.Country;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class CountryService {  
  
 public Country getCountry(String code) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 List<Country> countryList = (List<Country>) context.getBean("countryList");  
  
 return countryList.stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElse(null); // You can return a default or throw exception instead  
 }  
}

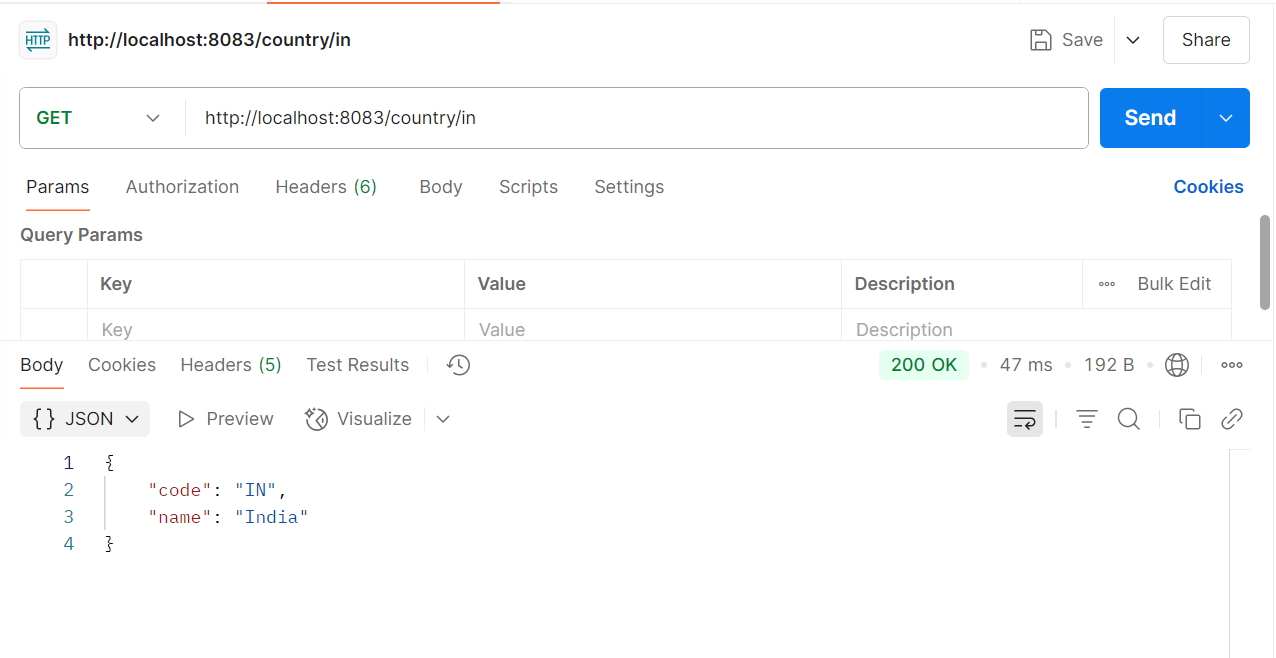
CountryController.java

package com.cognizant.spring\_learn.controller;  
  
import com.cognizant.spring\_learn.model.Country;  
import com.cognizant.spring\_learn.service.CountryService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
@RestController  
public class CountryController {  
  
 @Autowired  
 private CountryService countryService;  
  
 @GetMapping("/country/{code}")  
 public Country getCountry(@PathVariable String code) {  
 System.*out*.println("START");  
 Country country = countryService.getCountry(code);  
 System.*out*.println("END");  
 return country;  
 }  
}

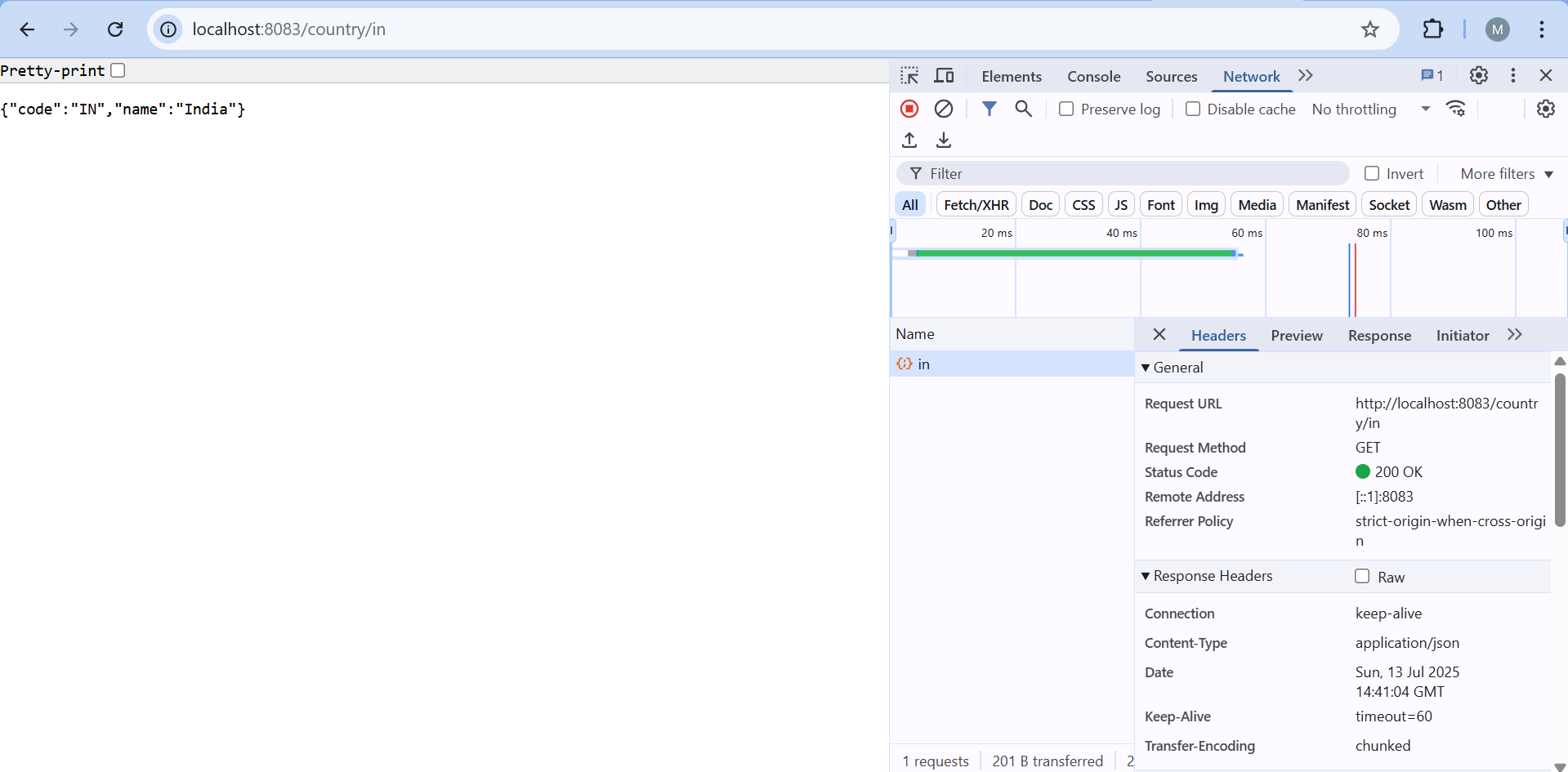
Testing the API in chrome

**

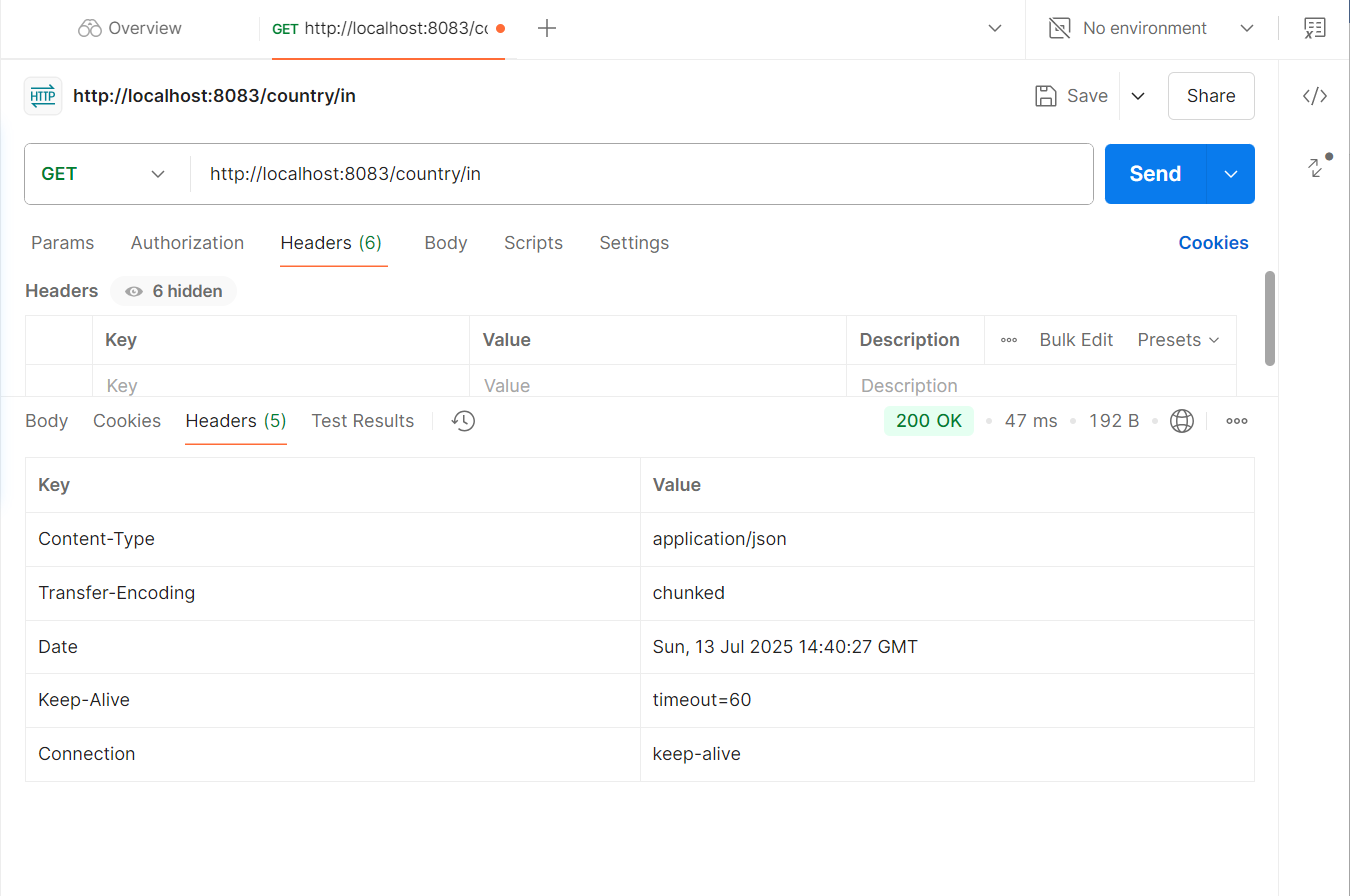
Testing the API in Postman



Chrome: View HTTP Headers



Postman: View HTTP Headers



*Hands On 6*

Create authentication service that returns JWT   
  
As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.  
  
**Request**

curl -s -u user:pwd http://localhost:8090/authenticate

**Response**

{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}

This can be incorporated as three major steps:

* Create authentication controller and configure it in SecurityConfig
* Read Authorization header and decode the username and password
* Generate token based on the user retrieved in the previous step

Let incorporate the above as separate hands on exercises.

*Solution*

Jwtauthapplication.java

package com.cognizant.jwt\_auth;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class JwtAuthApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JwtAuthApplication.class, args);  
 }  
  
}

AuthenticationController.java

package com.cognizant.jwt\_auth.controller;  
  
import com.cognizant.jwt\_auth.util.JwtUtil;  
import jakarta.servlet.http.HttpServletRequest;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Base64;  
  
@RestController  
public class AuthenticationController {  
  
 @Autowired  
 private JwtUtil jwtUtil;  
  
 @RequestMapping("/authenticate")  
 public ResponseEntity<?> generateToken(HttpServletRequest request) {  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader != null && authHeader.startsWith("Basic ")) {  
 String base64 = authHeader.substring(6); // Remove "Basic "  
 String decoded = new String(Base64.*getDecoder*().decode(base64));  
 String[] parts = decoded.split(":", 2);  
  
 if (parts.length == 2 && parts[0].equals("user") && parts[1].equals("pwd")) {  
 String token = jwtUtil.generateToken(parts[0]);  
 return ResponseEntity.*ok*("{\"token\":\"" + token + "\"}");  
 }  
 }  
  
 return ResponseEntity.*status*(401).body("Invalid credentials");  
 }  
}

JwtUtil.java

package com.cognizant.jwt\_auth.util;  
  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
  
@Component  
public class JwtUtil {  
  
 private final String secret = "secret-key"; // in production, store in env  
  
 public String generateToken(String username) {  
 return Jwts.*builder*()  
 .setSubject(username)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + 3600000)) // 1 hour  
 .signWith(SignatureAlgorithm.*HS256*, secret)  
 .compact();  
 }  
}

