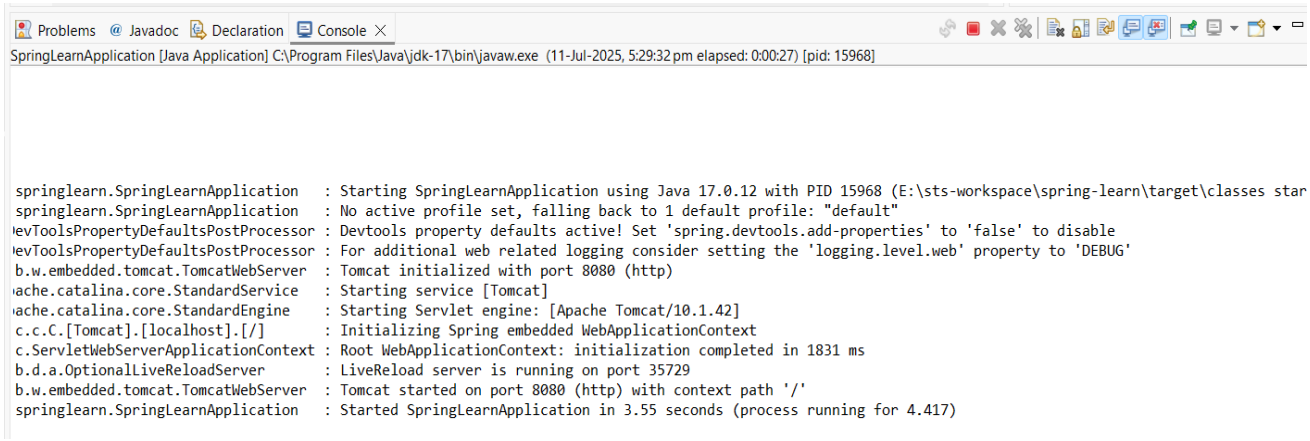
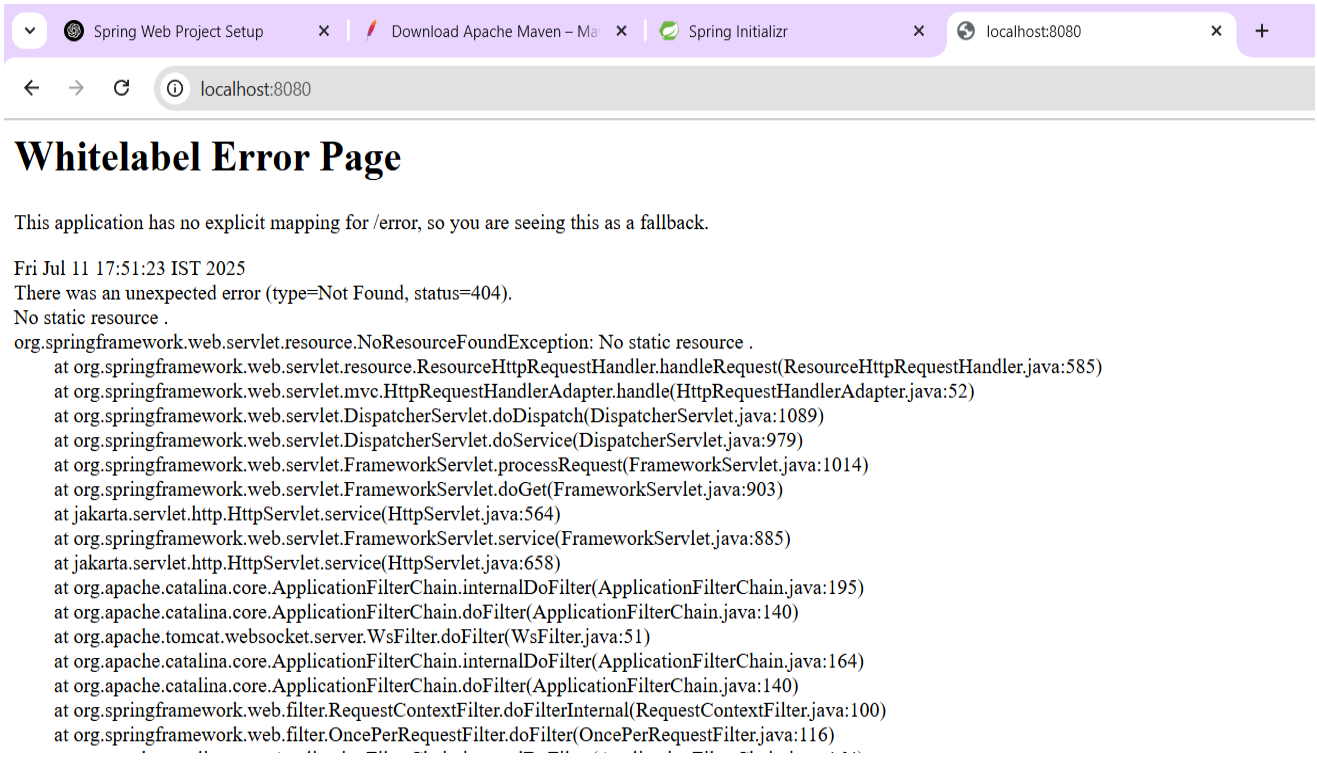
1. SPRING REST HANDSON

Hands on 1
Create a Spring Web Project using Maven

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/vafer/jdependency/2.10
 68 kB/s)
[INFO] Replacing main artifact E:\sts-workspace\spring-learn\target\spring-learn-0.0.1-S
ive, adding nested dependencies in BOOT-INF/.
[INFO] The original artifact has been renamed to E:\sts-workspace\spring-learn\target\sp
iginal
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  36.135 s
[INFO] Finished at: 2025-07-11T17:23:25+05:30
[INFO] ------------------------------------------------------------------------

E:\sts-workspace\spring-learn>
```





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

File name: Country.java

```
package com.cognizant.springlearn;

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 Constructor.");
   }

   public String getCode() {
      LOGGER.debug("Getting country code.");
      return code;
   }

   public void setCode(String code) {
      LOGGER.debug("Setting country code.");
      this.code = code;
   }

   public String getName() {
      LOGGER.debug("Getting country name.");
      return name;
   }

   public void setName(String name) {
      LOGGER.debug("Setting country name.");
      this.name = name;
   }

   @Override
   public String toString() {
      return "Country [code=" + code + ", name=" + name + "]";
   }
}
```

File 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
         http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="country" class="com.cognizant.springlearn.Country">
```

```
    <property name="code" value="IN" />
    <property name="name" value="India" />
  </bean>

</beans>
```

File name: Pom.xml

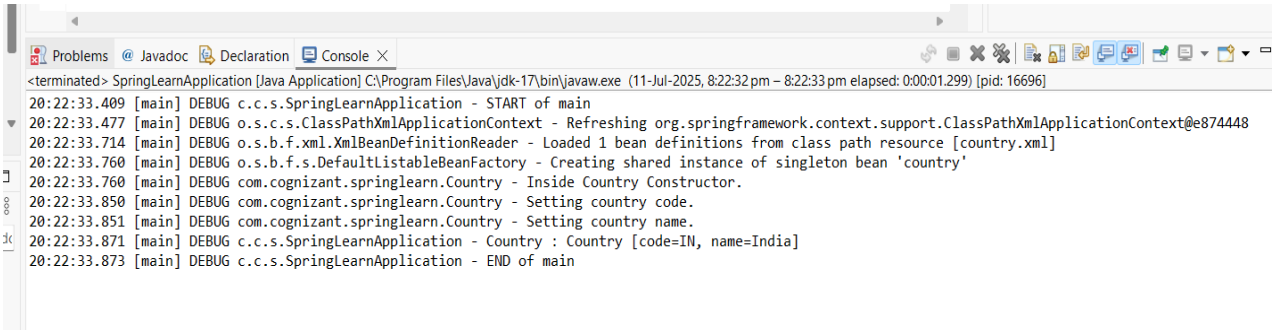
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
         http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.cognizant</groupId>
  <artifactId>spring-learn</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>spring-learn</name>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>

    <!-- Spring Core and Context -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.30</version>
    </dependency>

    <!-- SLF4J API for logging -->
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.7.36</version>
    </dependency>

    <!-- Logback Classic for SLF4J implementation -->
    <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
      <version>1.2.11</version>
    </dependency>

  </dependencies>
```

</project>

File name: SpringLearnApplication.java

```
package com.cognizant.springlearn;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

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

   public static void main(String[] args) {
      LOGGER.debug("START of main");
      displayCountry();
      LOGGER.debug("END of main");
   }

   public static void displayCountry() {
      ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

      Country country = context.getBean("country", Country.class);
      LOGGER.debug("Country : {}", country.toString());
   }
}
```

OUTPUT



<bean> Tag

Defines a bean (object) in the Spring container.

<bean id="country" class="com.cognizant.springlearn.Country">

- id -- unique name for the bean in Spring
- class -- full-qualified class name to be instantiated

<property> Tag

Used to set values of the bean's fields via setter methods.

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

- name matches the setter method name setCode()
- value is the string value passed

ApplicationContext

Interface that represents the Spring IoC (Inversion of Control) container.

It is responsible for:

- Loading bean definitions
- Instantiating and wiring beans
- Managing bean life cycle

ClassPathXmlApplicationContext

Implementation of ApplicationContext that loads context from an XML file located in the classpath.

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

This line tells Spring to:

- Look for country.xml in the classpath
- Parse it
- Instantiate all <bean>s defined in it

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

This retrieves the bean named "country" of type Country.

Internally:

1. Spring looks for a bean with id="country"
2. Uses reflection to instantiate the class com.cognizant.springlearn.Country
3. Calls its constructor
4. Invokes setter methods (setCode, setName) with values from XML
5. Returns the fully initialized Country object

2. SPRING REST HANDSON

Hello World RESTful Web Service

File name: HelloController.java

```
package com.cognizant.springlearn.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 response = "Hello World!!";

      LOGGER.info("END");

      return response;
```

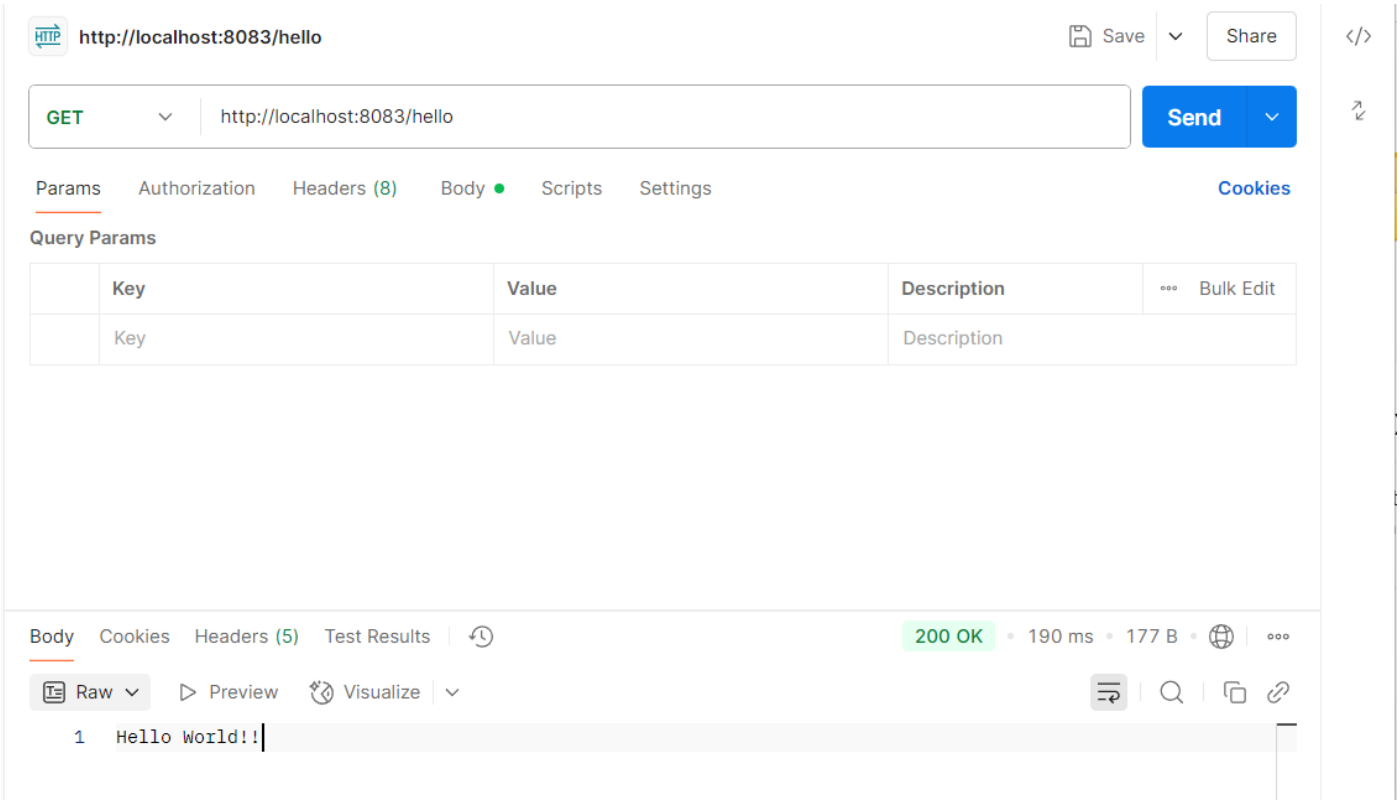
```
    }
}
```

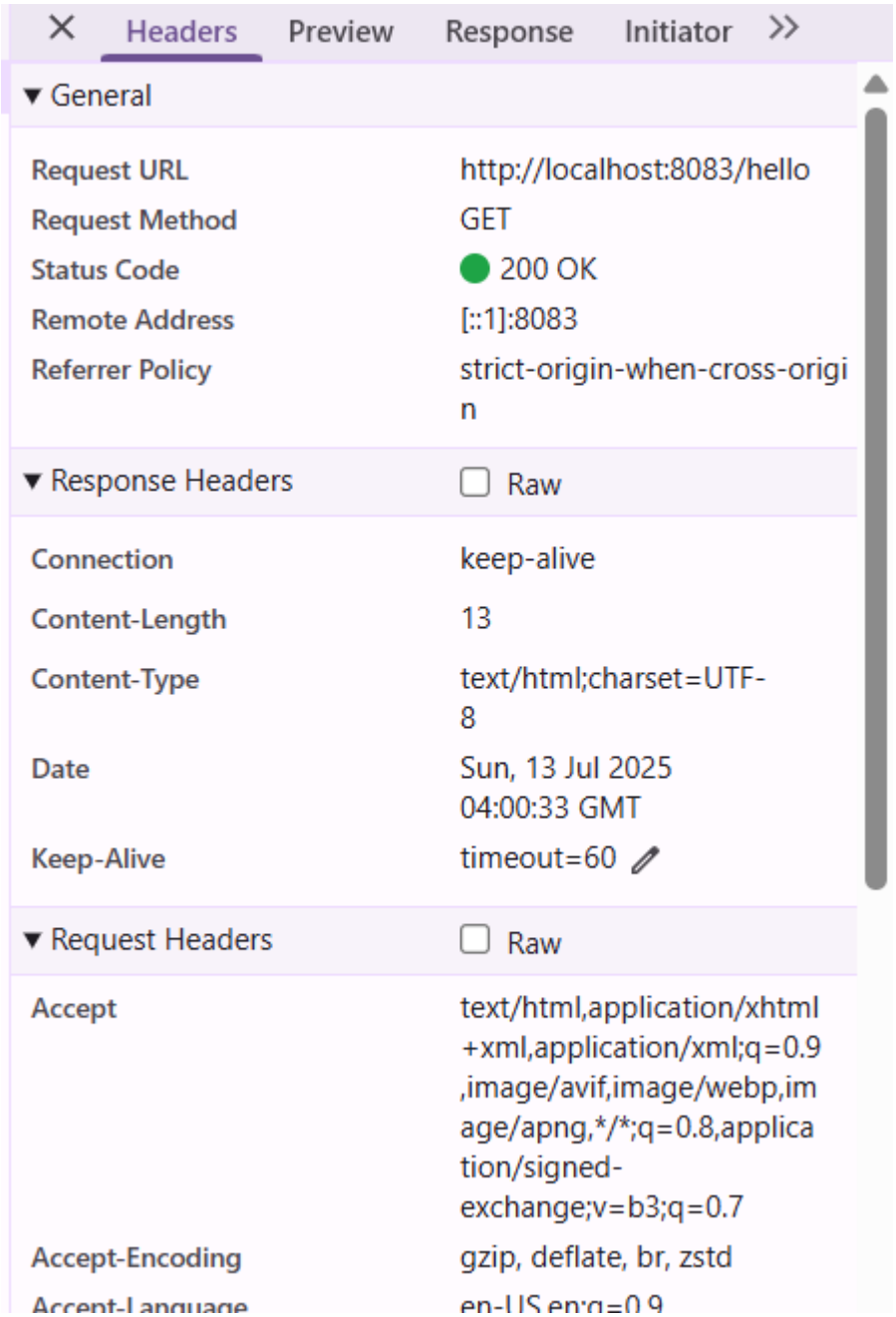
File name: SpringLearnApplication.java

```
package com.cognizant.springlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

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

    public static void main(String[] args) {

        LOGGER.debug("START of main");

        SpringApplication.run(SpringLearnApplication.class, args);

        LOGGER.debug("END of main");

    }

}
```

Output:

Hello World!!


http://localhost:8083/hello
Save
Share
GET
http://localhost:8083/hello
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
190 ms
177 B
Raw
Preview
Visualize
1
Hello World!!



Body Cookies Headers (5) Test Results 200 OK • 190 ms • 1

Key	Value
Content-Type	text/plain;charset=UTF-8
Content-Length	13
Date	Sun, 13 Jul 2025 03:53:37 GMT
Keep-Alive	timeout=60
Connection	keep-alive

REST - Country Web Service

File name: SpringLearnApplication.java

```
package com.cognizant.springlearn;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringLearnApplication {

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

    public static void main(String[] args) {
        LOGGER.debug("START of main");
        SpringApplication.run(SpringLearnApplication.class, args);
        LOGGER.debug("END of main");
    }
}
```

File name: Country.java

```
package com.cognizant.springlearn;

public class Country {
    private String code;
    private String name;

    public Country() {
        System.out.println("Inside Country Constructor.");
    }

    public String getCode() {
        return code;
    }
```

```
    public void setCode(String code) {
        System.out.println("Setting country code.");
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        System.out.println("Setting country name.");
        this.name = name;
    }

    @Override
    public String toString() {
        return "Country [code=" + code + ", name=" + name + "]";
    }
}
```

File name:Country.xml

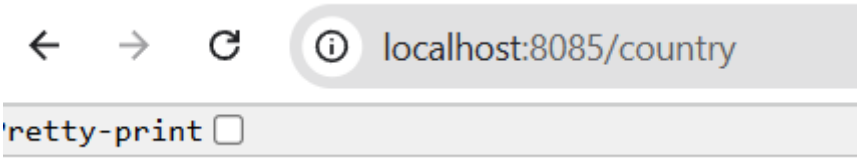
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                  http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="in" class="com.cognizant.springlearn.Country">
        <property name="code" value="IN"/>
        <property name="name" value="India"/>
    </bean>

</beans>
```

File name:CountryController.java

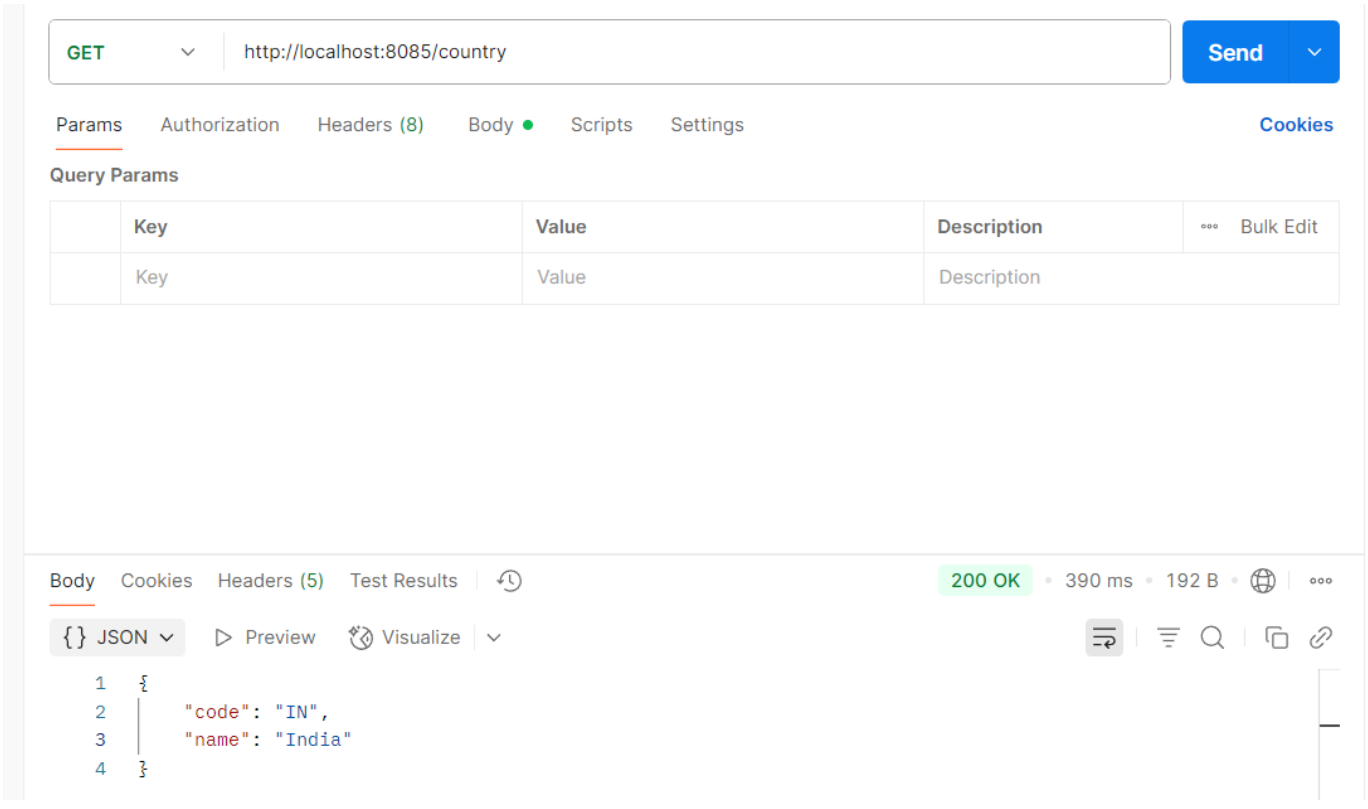
```
package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.Country;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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 {

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

   @RequestMapping("/country")
   public Country getCountryIndia() {
      LOGGER.info("START");

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

      LOGGER.info("END");
      return country;
   }
}
```

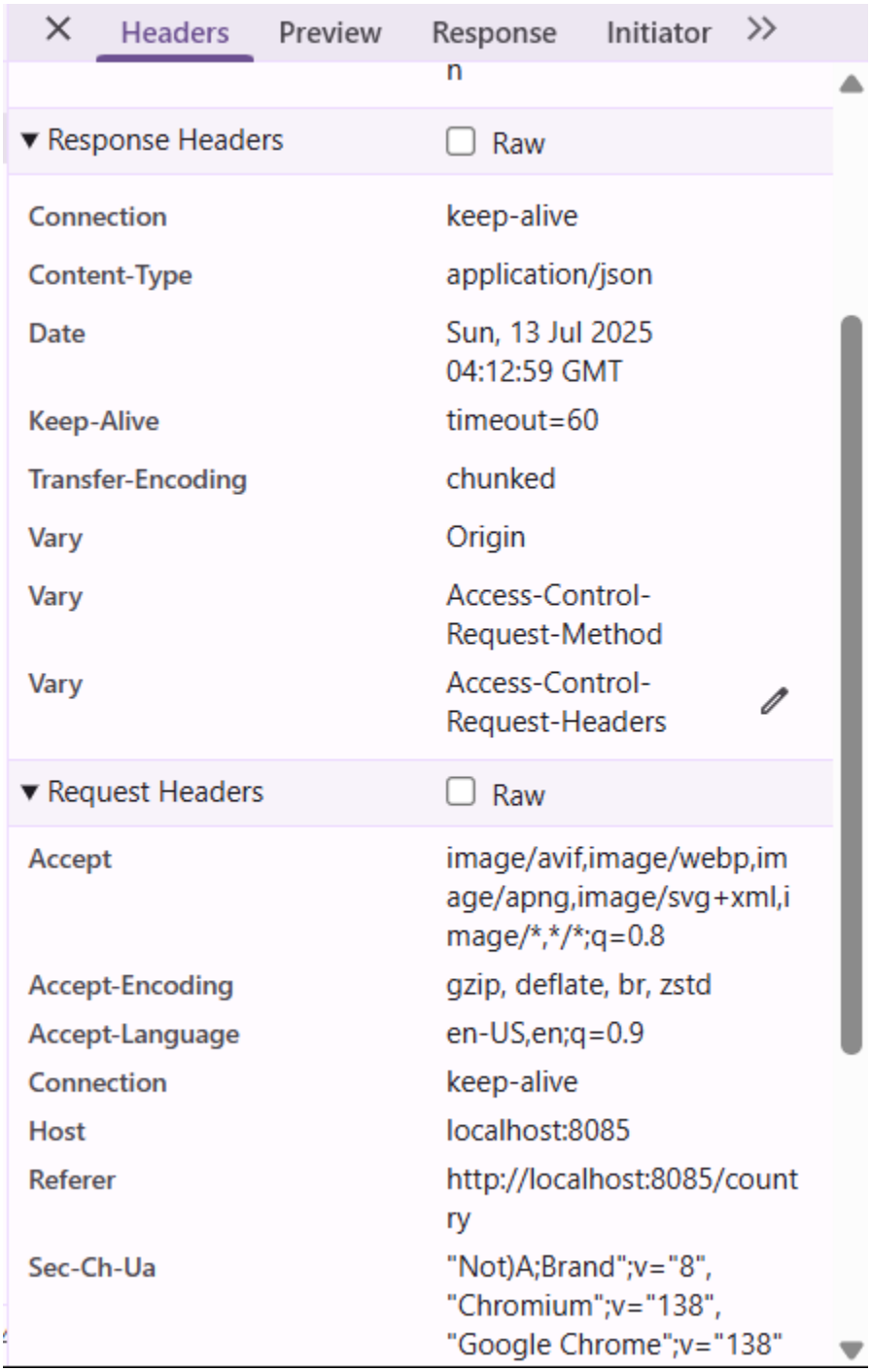
OUTPUT:

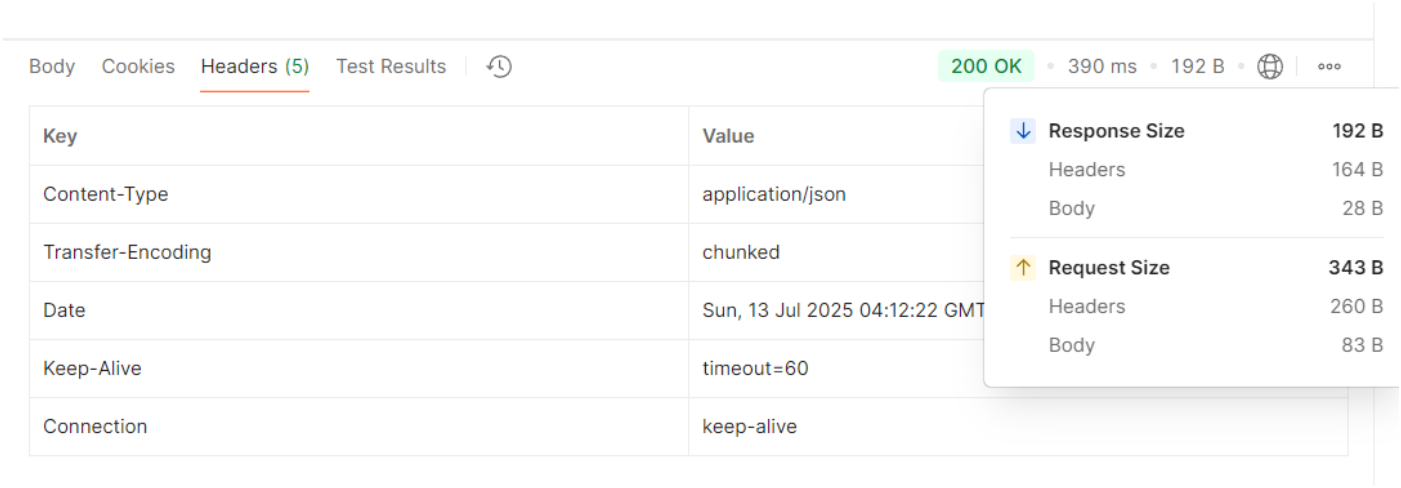
```
spring-learn - SpringLearnApplication [Spring Boot App] C:\Program Files\Java\jdk-17\bin\javaw.exe  (13-Jul-2025, 9:41:33 am elapsed: 0:00:26) [pid: 27256]
09:41:37.297 [restartedMain] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Autowiring by type from bean name 'optionalLiveReloadServer' via factory me
09:41:37.297 [restartedMain] DEBUG o.s.b.d.livereload.LiveReloadServer - Starting live reload server on port 35729
09:41:37.299 [restartedMain] INFO  o.s.b.d.a.OptionalLiveReloadServer - LiveReload server is running on port 35729
09:41:37.299 [restartedMain] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Creating shared instance of singleton bean 'liveReloadServerEventListener'
09:41:37.300 [restartedMain] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Autowiring by type from bean name 'liveReloadServerEventListener' via facto
09:41:37.300 [restartedMain] DEBUG o.s.b.f.s.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.boot.devtoo
09:41:37.309 [restartedMain] DEBUG o.s.j.e.a.AnnotationMBeanExporter - Registering beans for JMX exposure on startup
09:41:37.309 [restartedMain] DEBUG o.s.j.e.a.AnnotationMBeanExporter - Autodetecting user-defined JMX MBeans
09:41:37.319 [restartedMain] DEBUG o.s.c.s.DefaultLifecycleProcessor - Starting beans in phase -2147483647
09:41:37.319 [restartedMain] DEBUG o.s.c.s.DefaultLifecycleProcessor - Successfully started bean 'springBootLoggingLifecycle'
09:41:37.319 [restartedMain] DEBUG o.s.c.s.DefaultLifecycleProcessor - Starting beans in phase 2147481599
09:41:37.319 [restartedMain] INFO  o.a.coyote.http11.Http11NioProtocol - Starting ProtocolHandler ["http-nio-8085"]
09:41:37.355 [restartedMain] INFO  o.s.b.w.e.tomcat.TomcatWebServer - Tomcat started on port 8085 (http) with context path ''
09:41:37.357 [restartedMain] DEBUG o.s.c.s.DefaultLifecycleProcessor - Successfully started bean 'webServerStartStop'
09:41:37.358 [restartedMain] DEBUG o.s.c.s.DefaultLifecycleProcessor - Starting beans in phase 2147482623
09:41:37.358 [restartedMain] DEBUG o.s.c.s.DefaultLifecycleProcessor - Successfully started bean 'webServerGracefulShutdown'
09:41:37.358 [restartedMain] DEBUG o.s.c.s.DefaultLifecycleProcessor - Starting beans in phase 2147483647
09:41:37.375 [restartedMain] DEBUG o.s.b.a.l.ConditionEvaluationReportLogger -
```



"code":"IN","name":"India"}




Headers
Preview
Response
Initiator
n
Response Headers
Raw
Connection
keep-alive
Content-Type
application/json
Date
Sun, 13 Jul 2025 04:12:59 GMT
Keep-Alive
timeout=60
Transfer-Encoding
chunked
Vary
Origin
Vary
Access-Control-Request-Method
Vary
Access-Control-Request-Headers
Request Headers
Raw
Accept
image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Accept-Encoding
gzip, deflate, br, zstd
Accept-Language
en-US,en;q=0.9
Connection
keep-alive
Host
localhost:8085
Referer
http://localhost:8085/country
Sec-Ch-Ua
"Not)A;Brand";v="8", "Chromium";v="138", "Google Chrome";v="138"


Body
Cookies
Headers (5)
Test Results
200 OK
390 ms
192 B
Key
Value
Content-Type
application/json
Transfer-Encoding
chunked
Date
Sun, 13 Jul 2025 04:12:22 GMT
Keep-Alive
timeout=60
Connection
keep-alive
Response Size
192 B
Headers
164 B
Body
28 B
Request Size
343 B
Headers
260 B
Body
83 B

What happens in the controller method?

1. The controller receives a GET request on /country.
2. It logs START and loads Spring context from country.xml.
3. Retrieves the bean with id in — a Country object representing India.
4. Logs END and returns the bean as the HTTP response.

How is the bean converted into JSON?

- Spring Boot uses Jackson library (auto-included via spring-boot-starter-web) to convert Java objects (POJOs) into JSON format.
- The Country bean has getters, so Jackson serializes the object like this:

{ "code": "IN", "name": "India" }

- This is done automatically due to @RestController, which combines @Controller + @ResponseBody.

REST - Get country based on country code

File name:Country.java

```
package com.cognizant.springlearn;


public class Country {
   private String code;
   private String name;


   public Country() {}


   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;
   }
}
```

File name:Country.xml

```
<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
                http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="in" class="com.cognizant.springlearn.Country">
    <property name="code" value="IN"/>
    <property name="name" value="India"/>
  </bean>

  <bean id="us" class="com.cognizant.springlearn.Country">
    <property name="code" value="US"/>
    <property name="name" value="United States"/>
  </bean>

  <bean id="countryList" class="java.util.ArrayList">
    <constructor-arg>
      <list>
        <ref bean="in"/>
        <ref bean="us"/>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

File name:SpringLearnApplication.java

```
package com.cognizant.springlearn;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
```

```
public class SpringLearnApplication {

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

    public static void main(String[] args) {
        LOGGER.debug("START of main");
        SpringApplication.run(SpringLearnApplication.class, args);
        LOGGER.debug("END of main");
    }
}
```

File name:CountryController.java

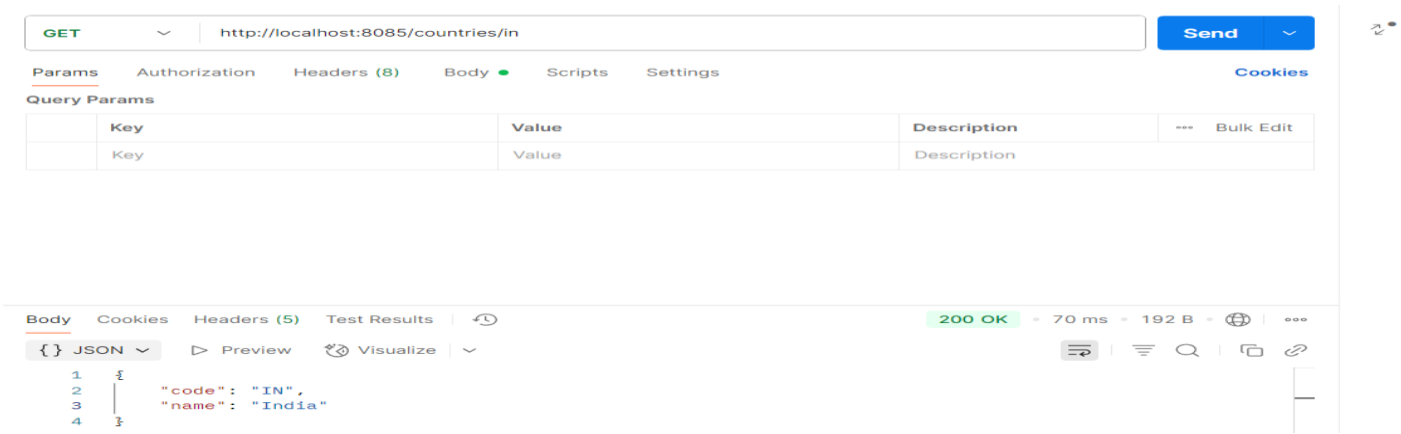
```
package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.Country;
import com.cognizant.springlearn.service.CountryService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
public class CountryController {

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

    @Autowired
    private CountryService countryService;

    @GetMapping("/countries/{code}")
    public Country getCountry(@PathVariable String code) {
        LOGGER.info("START getCountry with code: {}", code);
        Country result = countryService.getCountry(code);
        LOGGER.info("END");
        return result;
```

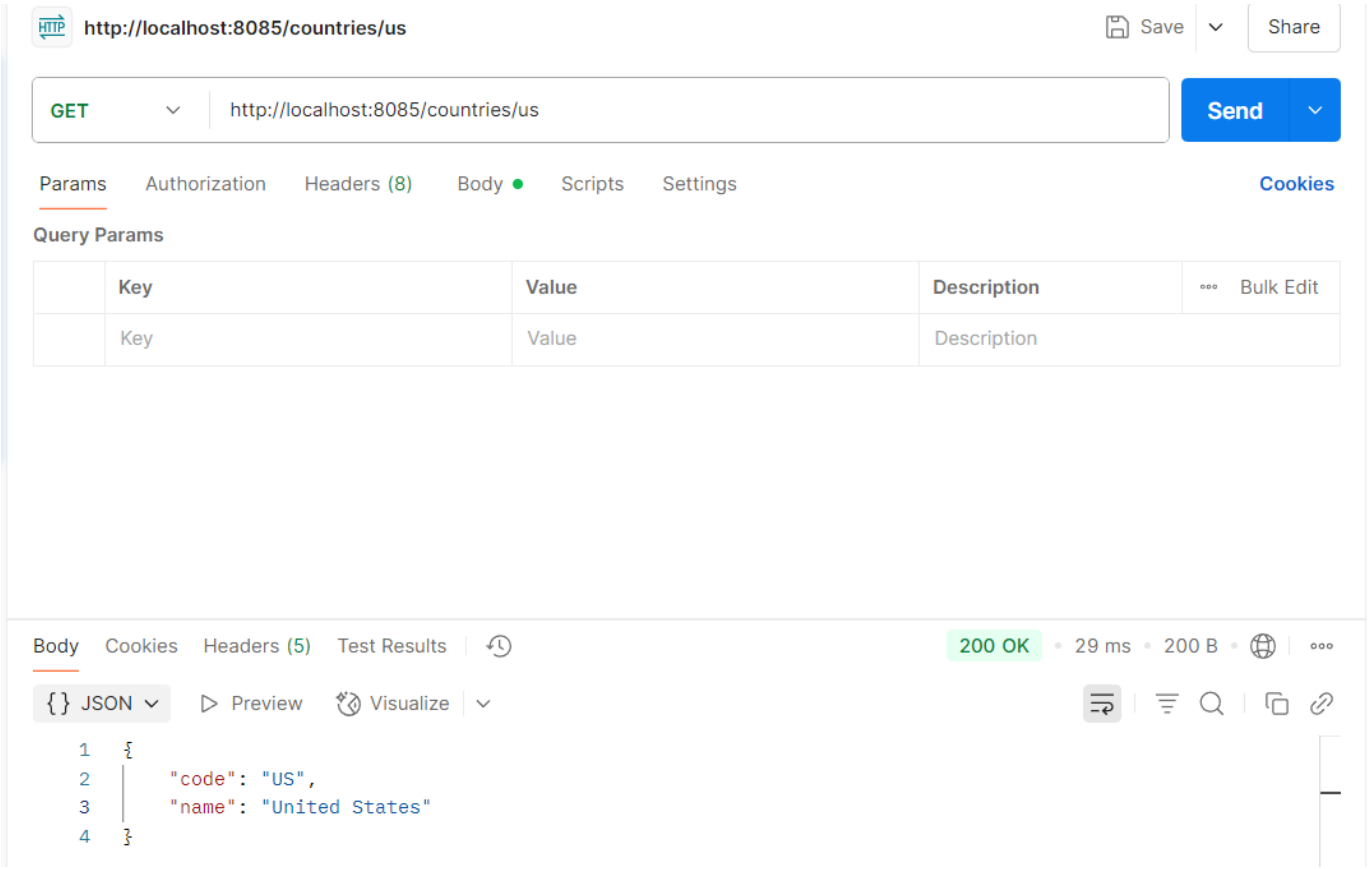
```
    }
}
```

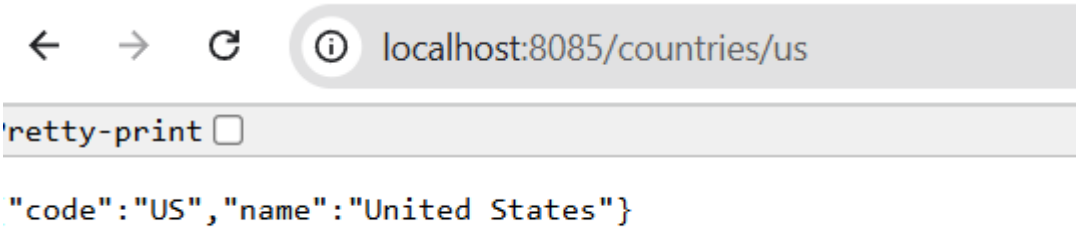
File name:CountryService.java

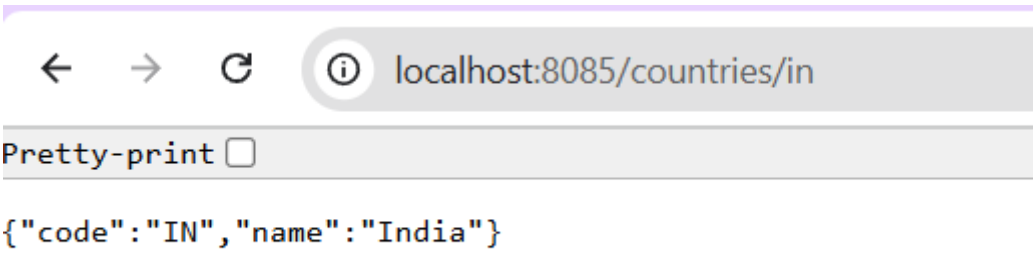
```
package com.cognizant.springlearn.service;

import com.cognizant.springlearn.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> countries = (List<Country>) context.getBean("countryList");

        // Find country ignoring case
        return countries.stream()
                .filter(c -> c.getCode().equalsIgnoreCase(code))
                .findFirst()
                .orElse(null); // or throw new CountryNotFoundException(code);
    }
}
```

Output:







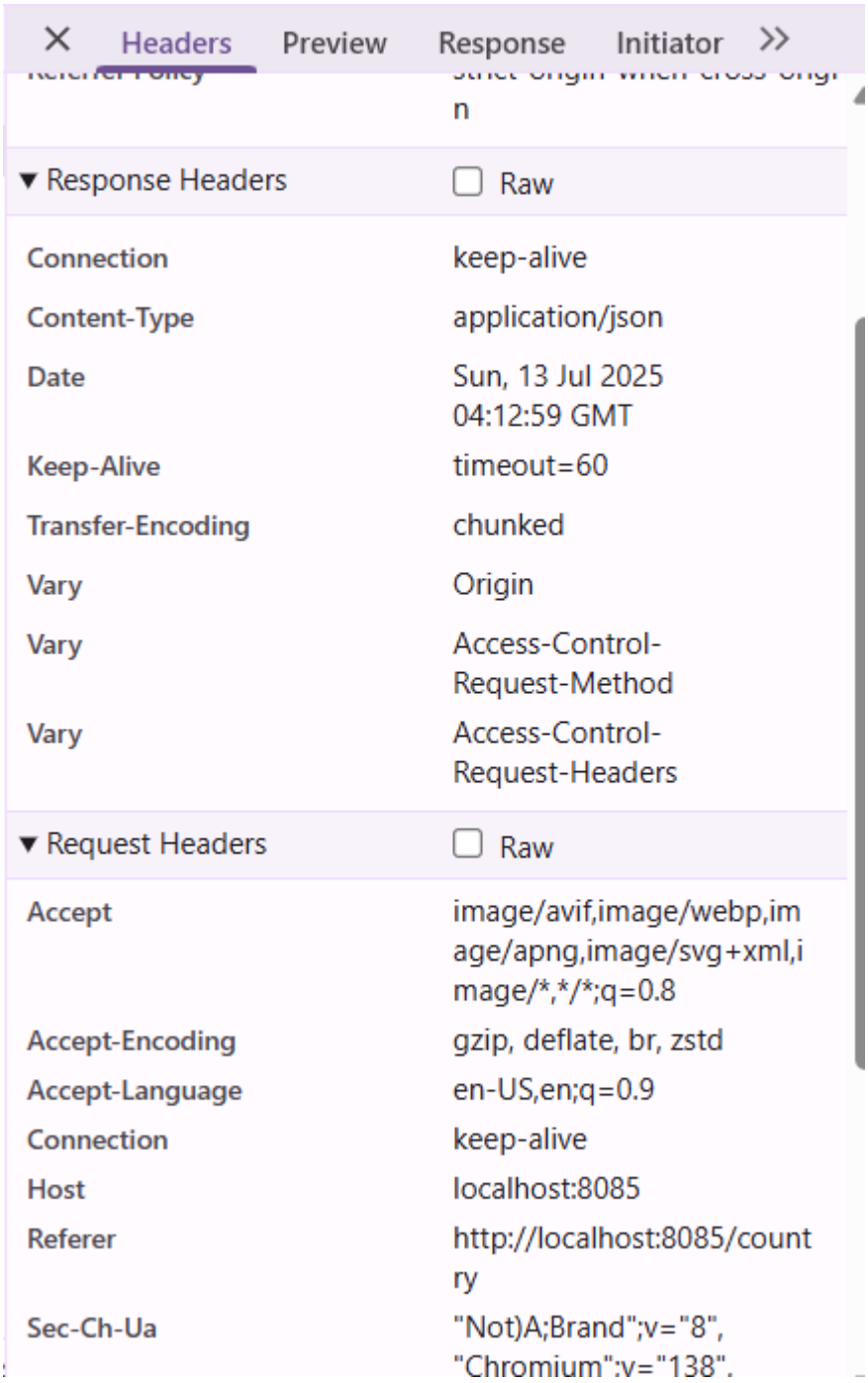


What happens in the controller?

- **Controller gets the {code} from the URL via @PathVariable**
- **It calls countryService.getCountry(code)**
- **The service fetches the countryList from Spring XML**
- **It performs a case-insensitive match and returns the matching country**

How is the bean converted to JSON?

- **Spring Boot uses Jackson (auto-configured with spring-boot-starter-web)**
- **Your Country class is a plain POJO with getters → automatically converted to JSON**



Body Cookies Headers (5) Test Results 200 OK • 29 ms • 200 B

Key	Value
Content-Type	application/json
Transfer-Encoding	chunked
Date	Sun, 13 Jul 2025 04:32:11 GMT
Keep-Alive	timeout=60
Connection	keep-alive

keep-alive

5. JWT-handson

Create authentication service that returns JWT

Create authentication controller and configure it in SecurityConfig

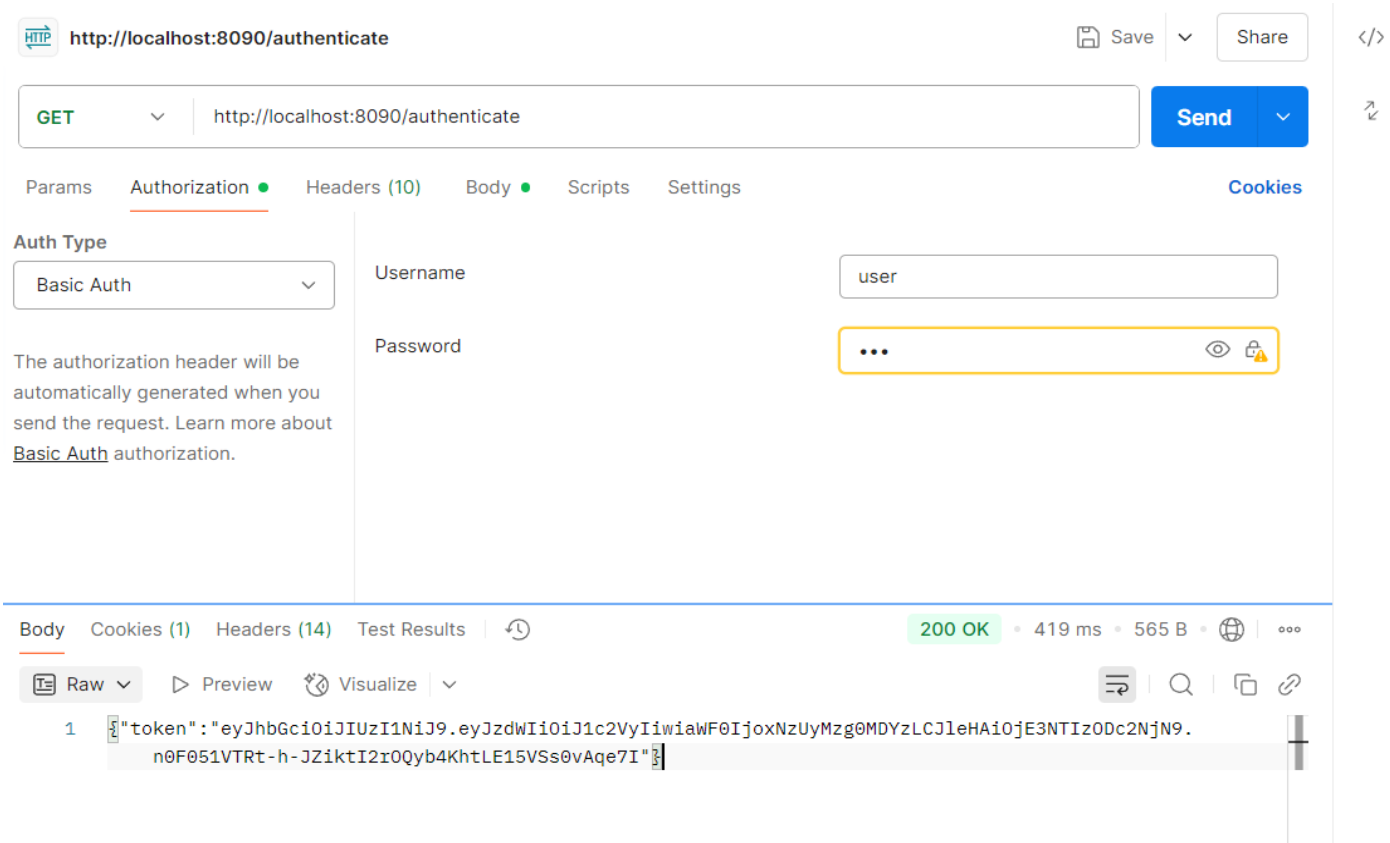
```
package com.cognizant.springlearn;

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);
   }
}
package com.cognizant.springlearn.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

   @Bean
   public InMemoryUserDetailsManager userDetailsService() {
      UserDetails user = User.builder()
         .username("user")
         .password("{noop}pwd") // {noop} = No password encoding
         .roles("USER")
```

```
            .build();
        return new InMemoryUserDetailsManager(user);
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/authenticate").permitAll()
                .anyRequest().authenticated()
            )
            .httpBasic();

        return http.build();
    }
}

package com.cognizant.springlearn.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.cognizant.springlearn.util.JwtUtil;

import java.util.Base64;

import jakarta.servlet.http.HttpServletRequest;

@RestController
public class AuthenticationController {

    @Autowired
    private JwtUtil jwtUtil;
```

```
    @GetMapping("/authenticate")
    public ResponseEntity<?> authenticate(HttpServletRequest request) {
        // Extract Authorization header
        String authHeader = request.getHeader("Authorization");

        if (authHeader == null || !authHeader.startsWith("Basic ")) {
            return ResponseEntity.status(401).body("Missing or invalid Authorization header");
        }

        // Decode Base64 credentials
        String base64Credentials = authHeader.substring("Basic ".length());
        String credentials = new String(Base64.getDecoder().decode(base64Credentials));
        String[] values = credentials.split(":", 2);

        String username = values[0];
        String password = values[1];
        System.out.println("🔐 Inside authenticate()");

        // For demo: accept only hardcoded user/pwd
        if ("user".equals(username) && "pwd".equals(password)) {
            String token = jwtUtil.generateToken(username);
            return ResponseEntity.ok().body("{\"token\":\"" + token + "\"}");
        } else {
            return ResponseEntity.status(403).body("Invalid credentials");
        }
    }
}
package com.cognizant.springlearn.util;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;
import org.springframework.stereotype.Component;
```

```
import javax.crypto.SecretKey;
import java.util.Date;

@Component
public class JwtUtil {

    private static final SecretKey SECRET_KEY = Keys.secretKeyFor(SignatureAlgorithm.HS256);
    private static final long EXPIRATION_TIME = 1000 * 60 * 60; // 1 hour

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
                .signWith(SECRET_KEY)
                .compact();
    }
}
```

OUTPUT



Read Authorization header and decode the username and password

File name: AuthenticationController.java

```
package com.cognizant.springlearn.controller;

import java.util.Base64;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.web.bind.annotation.*;

import com.cognizant.springlearn.util.JwtUtil;

@RestController
public class AuthenticationController {

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

   private final AuthenticationManager authenticationManager;
   private final JwtUtil jwtUtil;

   public AuthenticationController(AuthenticationManager authenticationManager, JwtUtil jwtUtil) {
      this.authenticationManager = authenticationManager;
      this.jwtUtil = jwtUtil;
   }

   @GetMapping("/authenticate")
   public ResponseEntity<?> authenticate(@RequestHeader(HttpHeaders.AUTHORIZATION) String authHeader) {
      LOGGER.debug("START of authenticate()");
```

```
    String username = getUser(authHeader);
    LOGGER.debug("Extracted Username: {}", username);

    try {
        // The actual authentication step (optional since JWT is already returned)
        Authentication auth = new UsernamePasswordAuthenticationToken(username, "pwd");
        authenticationManager.authenticate(auth);
    } catch (AuthenticationException e) {
        LOGGER.error("Authentication failed for user: {}", username);
        return ResponseEntity.status(401).build();
    }

    String token = jwtUtil.generateToken(username);
    LOGGER.debug("Generated JWT Token");

    LOGGER.debug("END of authenticate()");
    return ResponseEntity.ok().body("{\"token\":\"" + token + "\"}");
}

// 🔑 This is the method to decode and extract username from Base64 encoded Authorization header
private String getUser(String authHeader) {
    LOGGER.debug("START of getUser()");
    if (authHeader == null || !authHeader.startsWith("Basic ")) {
        LOGGER.error("Invalid Authorization header: {}", authHeader);
        return null;
    }

    String encodedCredentials = authHeader.substring("Basic ".length());
    byte[] decodedBytes = Base64.getDecoder().decode(encodedCredentials);
    String decodedCredentials = new String(decodedBytes); // will be user:pwd

    LOGGER.debug("Decoded credentials: {}", decodedCredentials);
```

```
        String username = decodedCredentials.split(":")[0];


        LOGGER.debug("Extracted Username from Authorization header: {}", username);
        LOGGER.debug("END of getUser()");
        return username;
    }
}
```

OUTPUT: