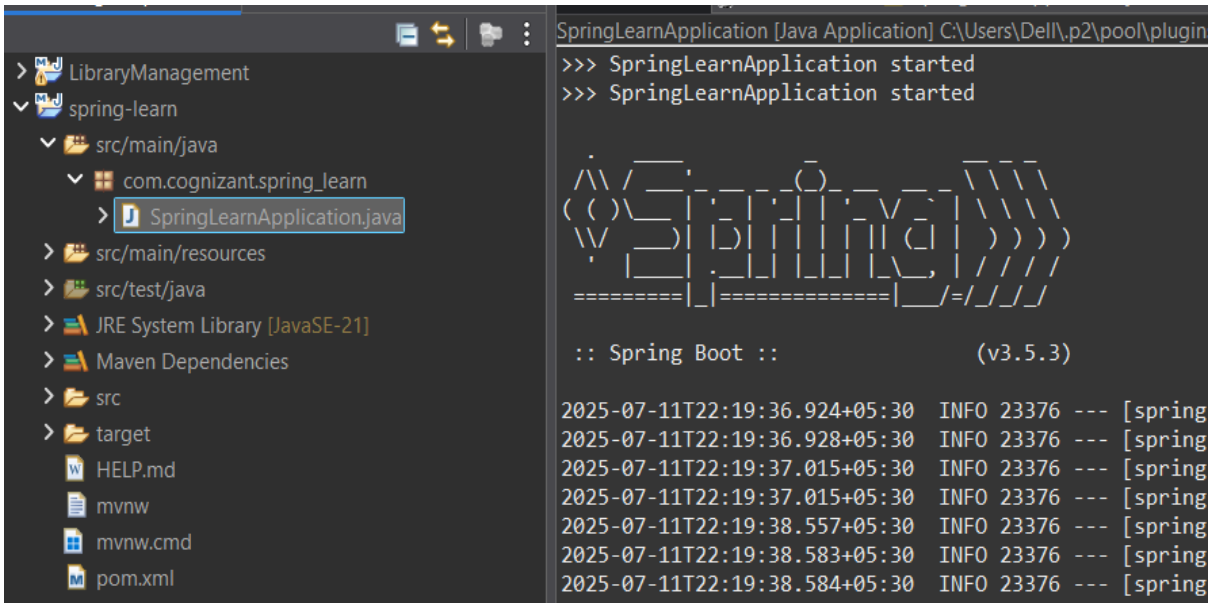
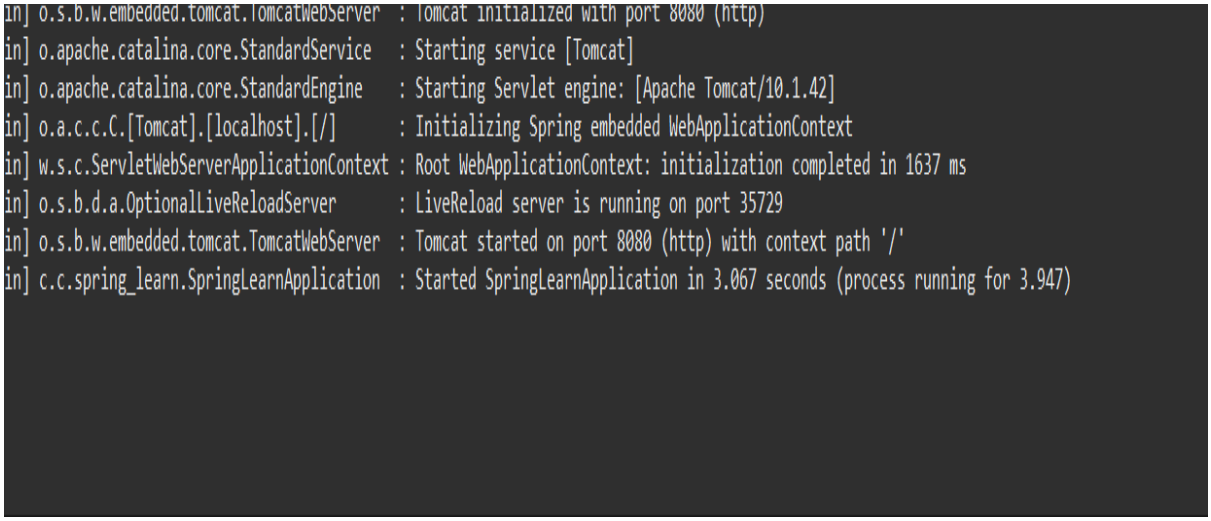
## Create a Spring Web Project using Maven

SpringLearnApplication 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) {
		System.out.println(">>> SpringLearnApplication started");
	  SpringApplication.run(SpringLearnApplication.class, args);
	}

}
```

**OUTPUT:**



## Spring Core – Load Country from Spring Configuration XML

**SpringLearnApplication Code:**

```
package com.cognizant.spring_learn;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringLearnApplication implements CommandLineRunner {

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

  public static void main(String[] args) {
    SpringApplication.run(SpringLearnApplication.class, args);
  }

  @Override
  public void run(String... args) {
    LOGGER.debug("START");
    displayCountry();
    LOGGER.debug("END");
  }

  public static void displayCountry() {
    ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
    Country country = context.getBean("country", Country.class);
    LOGGER.debug("Country : {}", country.toString());
  }
}
```

**Country.java**

```
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 Constructor.");
```

```
    }
    public String getCode() {
        LOGGER.debug("getCode() called");
        return code;
    }
    public void setCode(String code) {
        LOGGER.debug("setCode() called with: {}", code);
        this.code = code;}
    public String getName() {
        LOGGER.debug("getName() called");
        return name;
    }
    public void setName(String name) {
        LOGGER.debug("setName() called with: {}", name);
        this.name = name;
    }
    @Override
    public String toString() {
        return "Country [code=" + code + ", name=" + name + "]";
    }
}
```

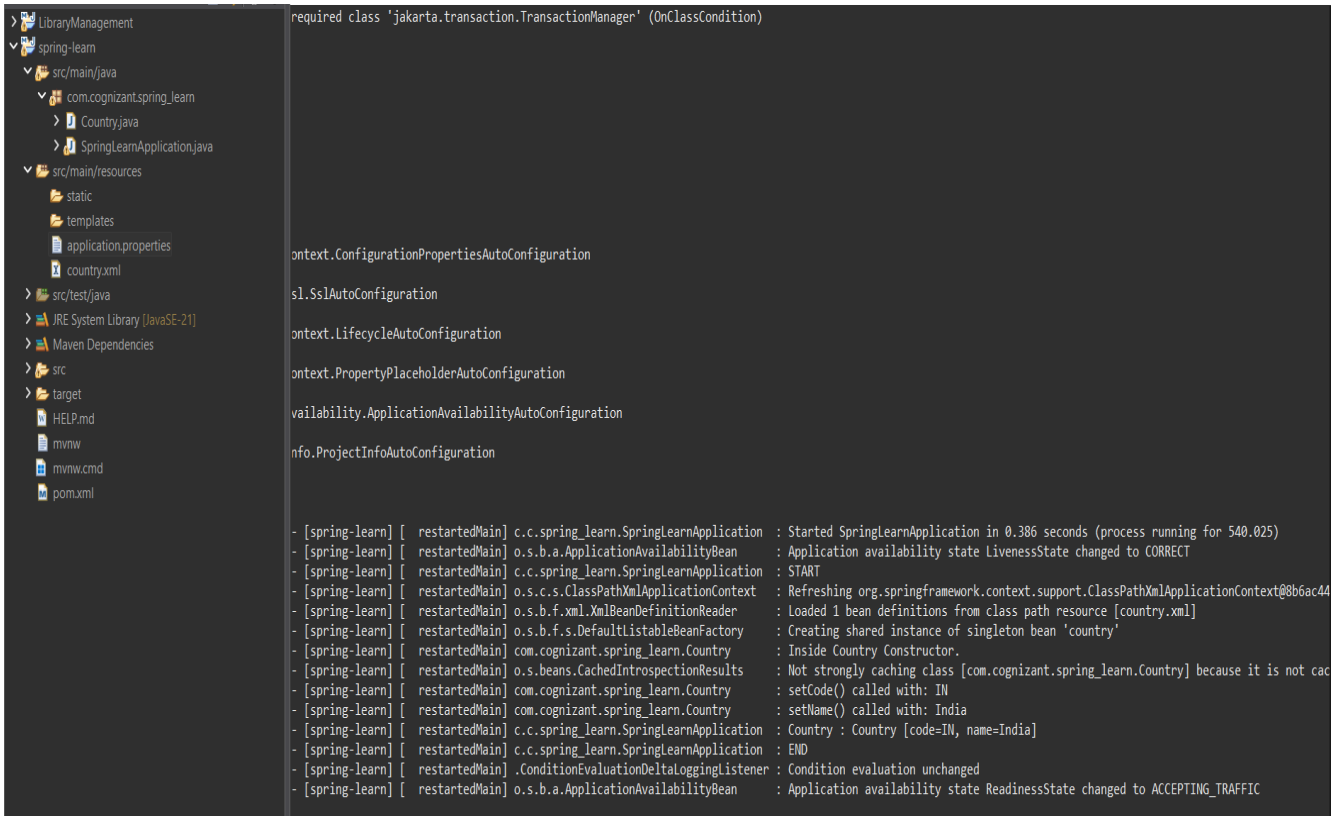
**Application.properties**

```
spring.application.name=spring-learn

logging.level.root=DEBUG
```

**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="country" class="com.cognizant.spring_learn.Country">
        <property name="code" value="IN" />
        <property name="name" value="India" />
    </bean>
</beans>
```

**OUTPUT:**




## Hello World RESTful Web Service

**STATEMENT:**

**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!!

**CODE:**

**HelloController.java:**

```
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.debug("START: sayHello()");
    String response = "Hello World!!";
    LOGGER.debug("END: sayHello()");
    return response;
  }
}
```

### SME Explanation

**Developer Tools (Chrome)**

1. Open Chrome → Visit http://localhost:8083/hello
2. Press F12 → Go to **Network** tab
3. Reload the page
4. Click on the /hello request
5. View these **Response Headers**:
    - Content-Type: text/plain;charset=UTF-8
    - Content-Length: 14
    - Date: ...
    - Connection: keep-alive

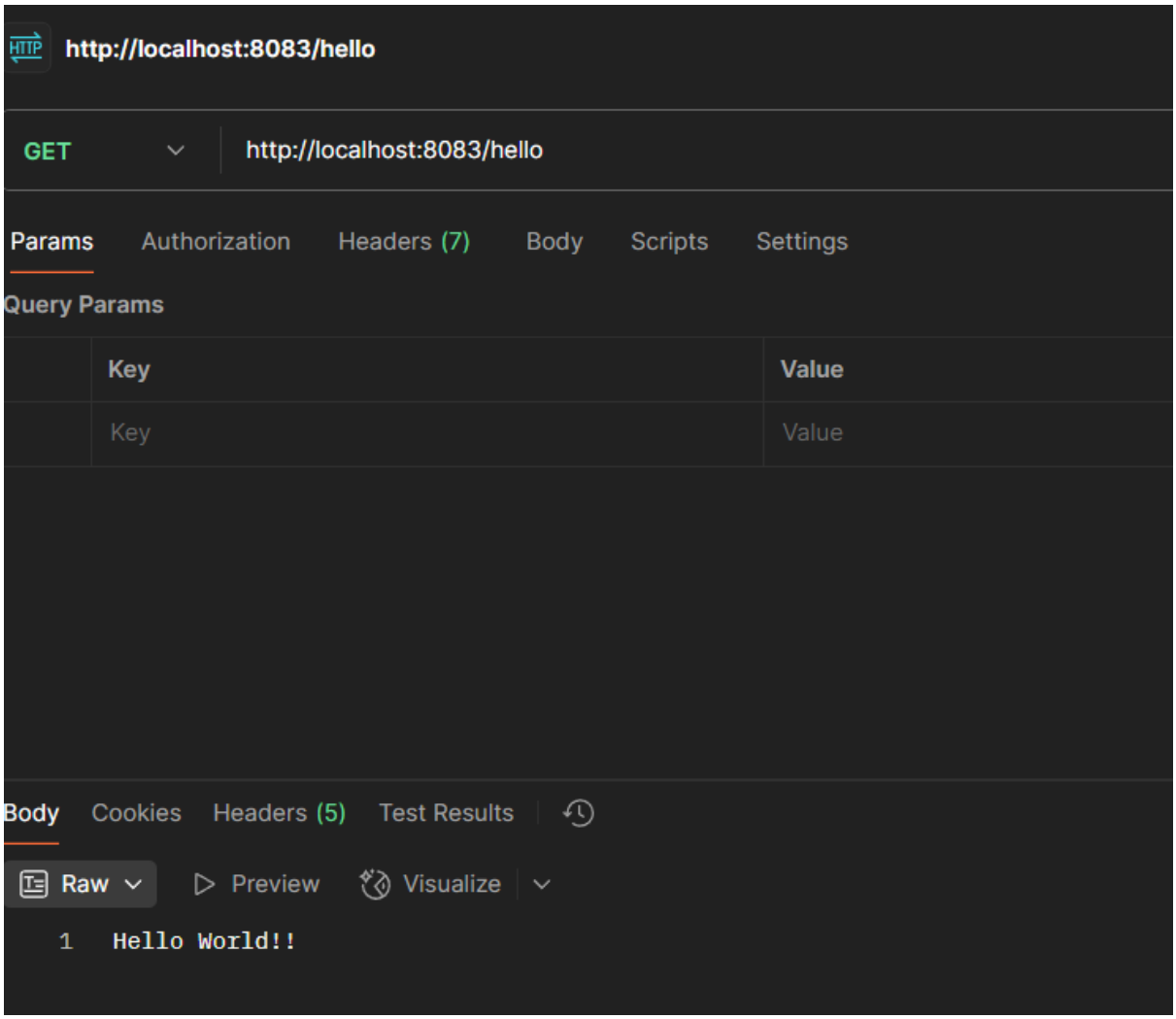
Also view **Request Headers**:

- GET /hello HTTP/1.1
- Host: localhost:8083
- User-Agent: Mozilla/...
- Accept: text/html,...

**Postman → Headers Tab**

1. After sending GET request to /hello, click on the **Headers** tab.
2. You'll see similar headers:
    - **Response Headers**:
        - Content-Type: text/plain;charset=UTF-8
        - Date, Connection, Content-Length, etc.
    - **Request Headers**:
        - User-Agent, Accept, Host

**OUTPUT:**


http://localhost:8083/hello
GET
http://localhost:8083/hello
Params
Authorization
Headers (7)
Body
Scripts
Settings
Query Params
Key
Value
Key
Value
Body
Cookies
Headers (5)
Test Results
Raw
Preview
Visualize
1 Hello World!!


Body
Cookies
Headers (5)
Test Results
200 OK
578 ms
177 B
Key
Value
Content-Type
text/plain;charset=UTF-8
Content-Length
13
Date
Sat, 12 Jul 2025 16:14:50 GMT
Keep-Alive
timeout=60
Connection
keep-alive

## REST - Country Web Service

**STATEMENTS:**

**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:**

**CountryController.java**

```
package com.cognizant.spring_learn.controller;
import com.cognizant.spring_learn.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.debug("START: getCountryIndia()");
    ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
    Country country = context.getBean("country", Country.class);
    LOGGER.debug("END: getCountryIndia()");
    return country;
  }
}
```

## SME:

**What happens in the controller method?**

```
@RequestMapping("/country")

public Country getCountryIndia() {

  ...

}
```

- This method handles HTTP **GET** requests to /country.
- It loads the Spring bean (Country) from country.xml using ApplicationContext.
- Returns a **Java object**, not a JSON string — Spring takes care of conversion.

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

1. Spring Boot includes **Jackson** (a JSON processor) in its dependencies.
2. When a Java object is returned from a @RestController method, Spring uses Jackson to automatically:
    - Inspect the object's fields via getters
    - Convert it to a JSON string
    - Set Content-Type: application/json in the response headers

**In Chrome Developer Tools**

1. Visit http://localhost:8083/country
2. Press F12 → Go to **Network** tab → Reload
3. Click the /country request

**Request Headers:**

- GET /country
- Host: localhost:8083
- Accept: application/json

**In Postman → Headers Tab**

1. Send a **GET** request to http://localhost:8083/country
2. Click the **Headers** tab

http://localhost:8083/country

GET http://localhost:8083/country

Params Authorization Headers (7) Body Scripts Settings

Query Params

| | Key | Value |
|---|---|---|
| | Key | Value |

Body Cookies Headers (5) Test Results

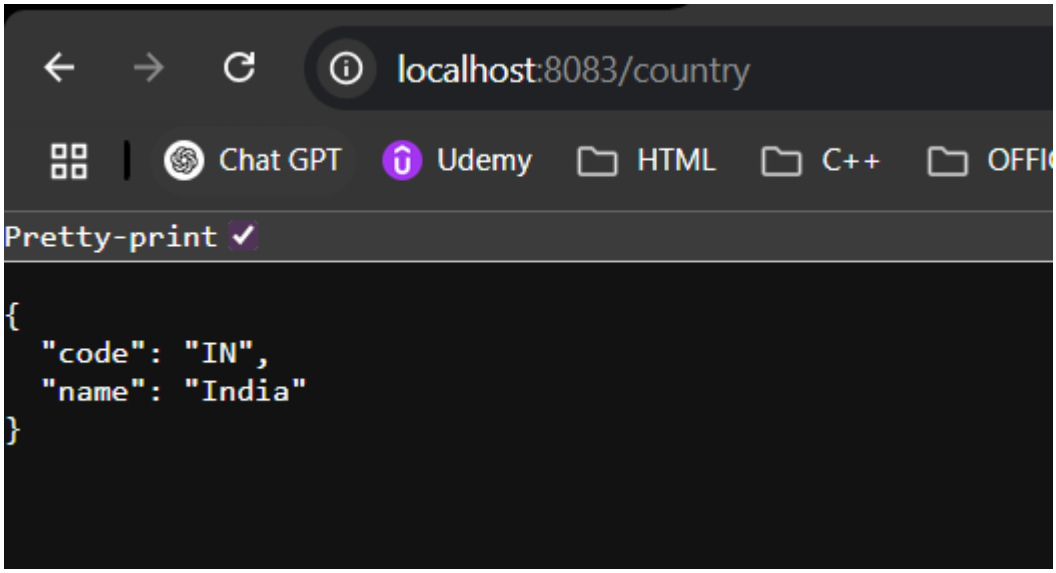
{} JSON Preview Visualize


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


:

Body Cookies Headers (5) Test Results 200 OK 151 ms 192 B

| Key | Value |
|---|---|
| Content-Type | application/json |
| Transfer-Encoding | chunked |
| Date | Sat, 12 Jul 2025 16:54:24 GMT |
| Keep-Alive | timeout=60 |
| Connection | keep-alive |



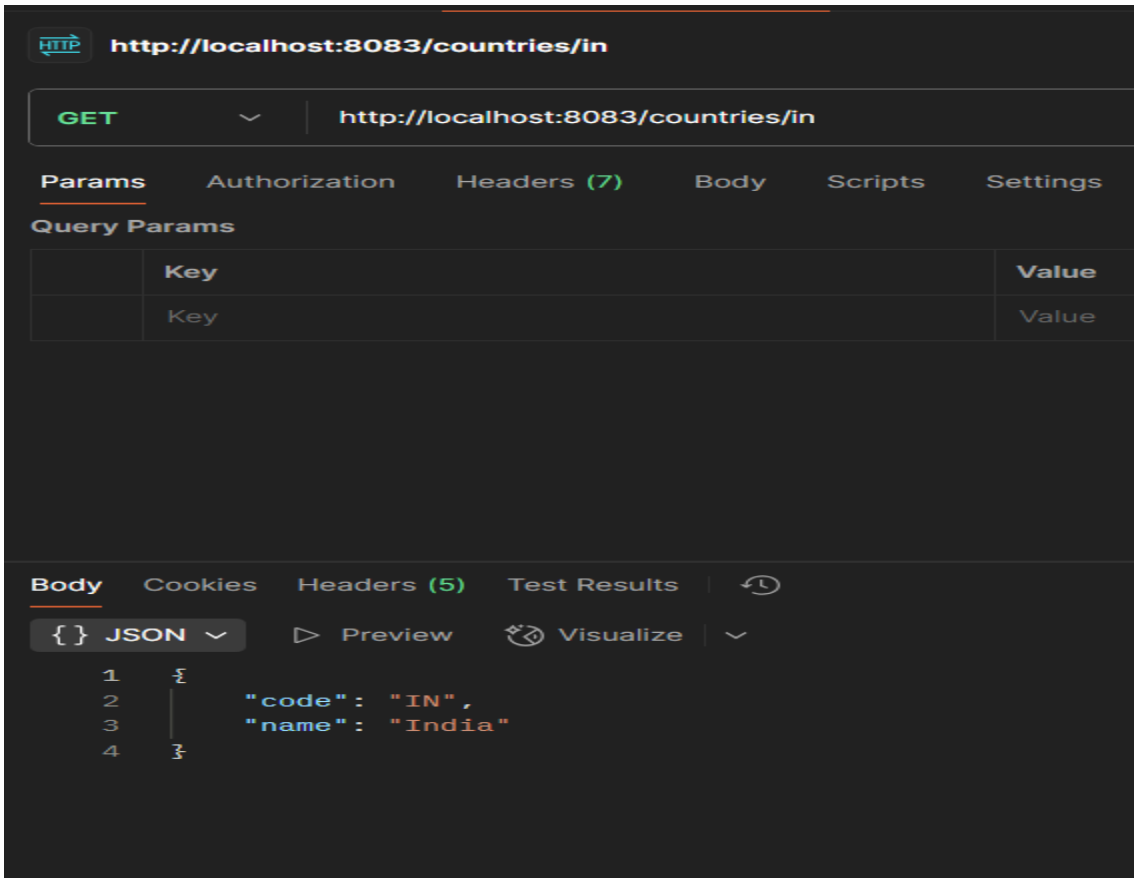

## REST - Get country based on country code

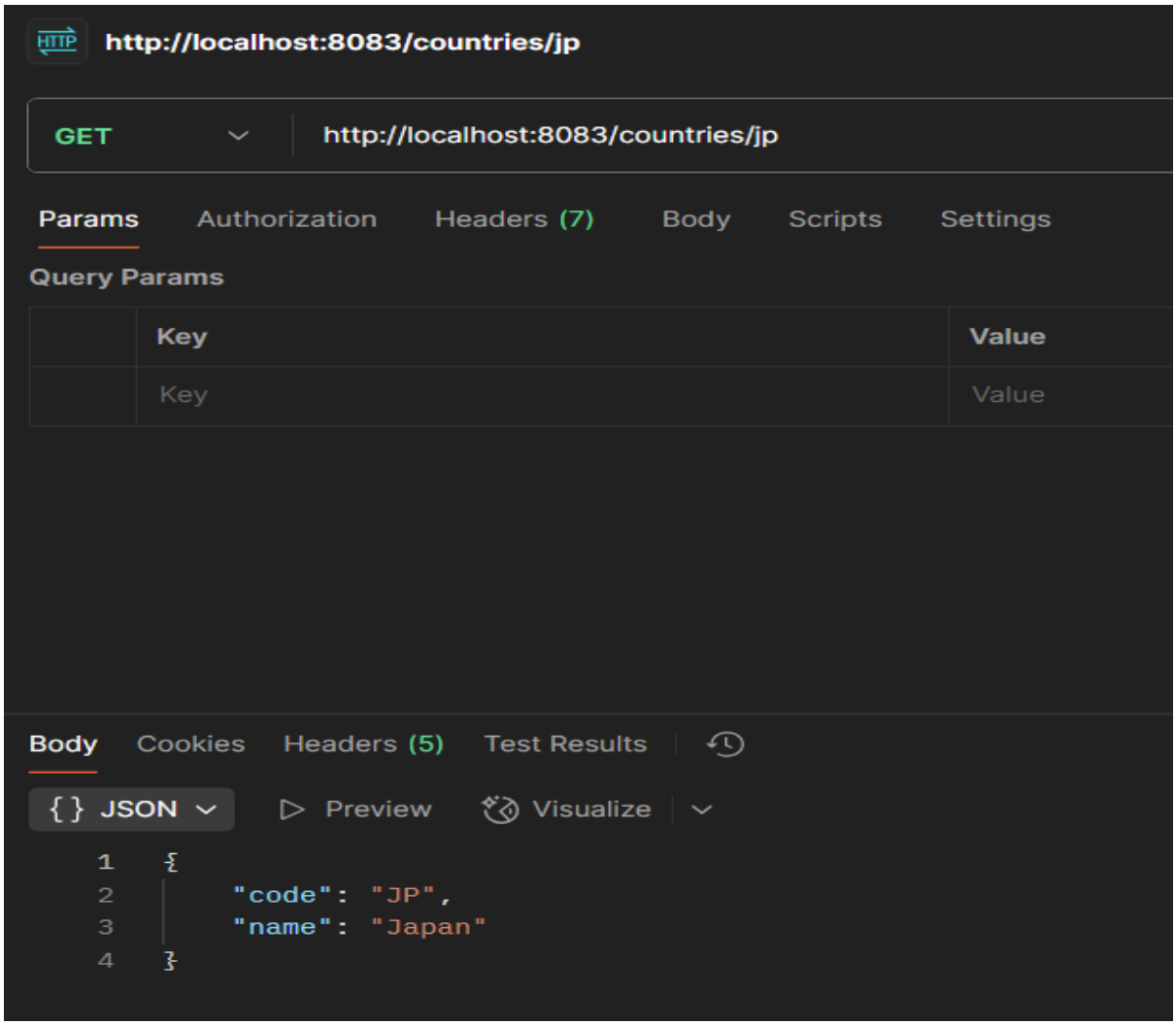
**CountryController.java**

```
package com.cognizant.spring_learn.controller;

import com.cognizant.spring_learn.Country;
import com.cognizant.spring_learn.service.CountryService;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@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.debug("START: getCountry()");
    Country country = countryService.getCountry(code);
    LOGGER.debug("END: getCountry()");
    return country;
  }
}
```

**CountryService.java**

```
package com.cognizant.spring_learn.service;

import com.cognizant.spring_learn.Country;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class CountryService {
  private static final Logger LOGGER = LoggerFactory.getLogger(CountryService.class);
  public Country getCountry(String code) {
    LOGGER.debug("Looking for country code: {}", code);
    ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
    List<Country> countries = (List<Country>) context.getBean("countryList");
    LOGGER.debug("Total countries found in config: {}", countries.size());
    return countries.stream()
        .filter(c -> c.getCode().equalsIgnoreCase(code))
        .findFirst()
        .orElse(null);
  }
}
```






## Create authentication service that returns JWT

## Exercise:

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.

**CODE:**

**AuthController.java**

```
package com.cognizant.spring_learn.controller;

import com.cognizant.spring_learn.util.JwtUtil;

import jakarta.servlet.http.HttpServletRequest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Base64;

@RestController
public class AuthController {

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

  @GetMapping("/authenticate")
  public ResponseEntity<?> authenticate(HttpServletRequest request) {
    LOGGER.debug("Authenticating...");

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

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

    String username = values[0];
    String password = values[1];

    // Hardcoded user credentials
    if ("user".equals(username) && "pwd".equals(password)) {
      String token = JwtUtil.generateToken(username);
      return ResponseEntity.ok().body("{\"token\":\"" + token + "\"}");
    } else {
      return ResponseEntity.status(401).body("Invalid credentials");
    }
  }
}
```

**Pom.xml**

**Add the following dependencies in the project**

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-security -->
<dependency>
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>com.auth0</groupId>
  <artifactId>java-jwt</artifactId>
  <version>4.4.0</version>
```

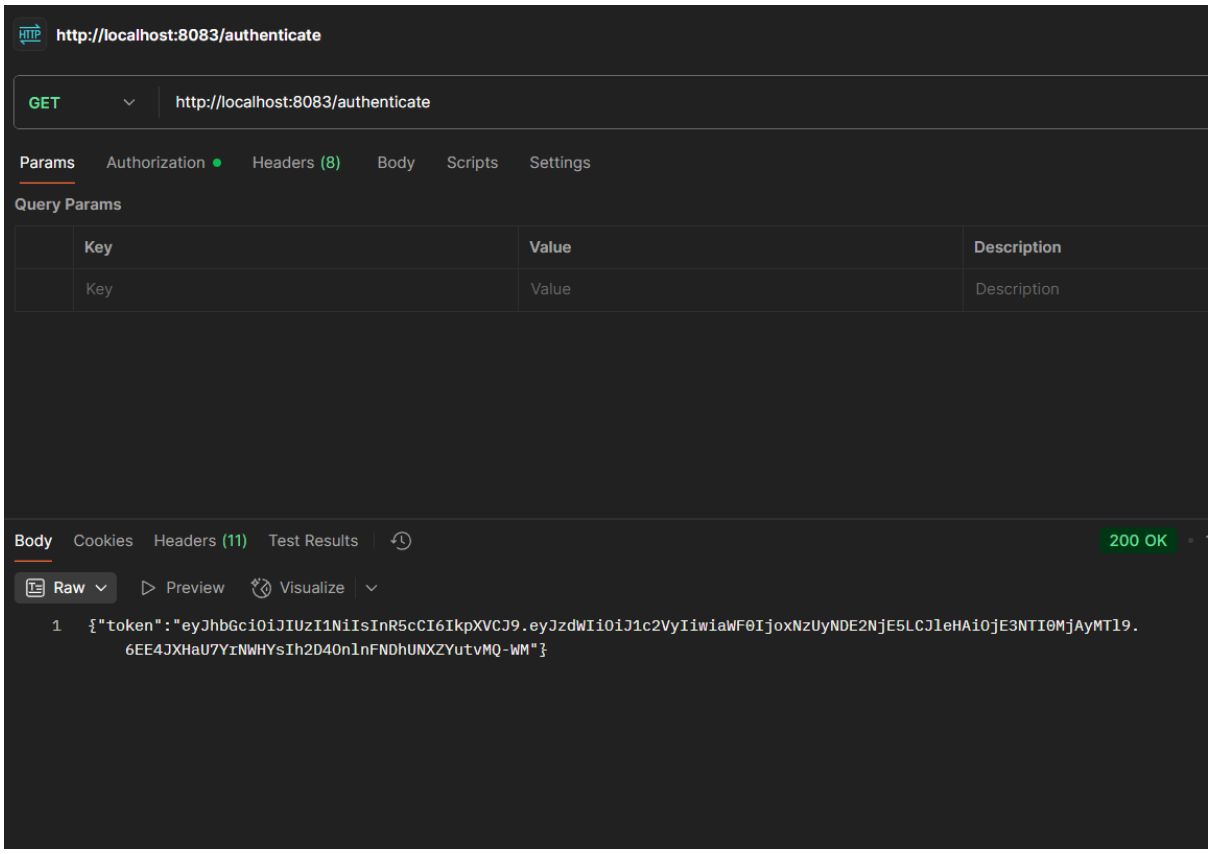
**SecurityConfig.java**

```
package com.cognizant.spring_learn.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
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.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {
        @Bean
        public UserDetailsService userDetailsService() {
          UserDetails user = User.builder()
            .username("user")
            .password(passwordEncoder().encode("pwd"))
            .roles("USER")
            .build();

          return new InMemoryUserDetailsManager(user);
        }

        @Bean
        public PasswordEncoder passwordEncoder() {
          return new BCryptPasswordEncoder();
        }
        @Bean
        public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
          http
            .authorizeHttpRequests(auth -> auth
              .requestMatchers("/authenticate").permitAll()
              .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults())
            .csrf(csrf -> csrf.disable());

          return http.build();
```

```
        }

}
```

**JwtUtil.java**

```
package com.cognizant.spring_learn.util;

import java.util.Date;

import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;

public class JwtUtil {
  public static String generateToken(String username) {
    Algorithm algorithm = Algorithm.HMAC256("secret1234567890123456789012 34");
    return JWT.create()
      .withSubject(username)
      .withIssuedAt(new Date())
      .withExpiresAt(new Date(System.currentTimeMillis() + 3600000))
      .sign(algorithm);
  }
}
```



**FILE STRUCTURE:**