**Spring Rest Hands - on**

**Objective**

**1. Demonstrate Creation of Spring Boot Application**

A Spring Boot application simplifies Java application development using the Spring framework. Here's how you can create one:

**Steps to Create a Spring Boot Application:**

1. Go to [Spring Initializr](https://start.spring.io)
2. Fill in:
   * Group: com.example
   * Artifact: demo
   * Dependencies: Spring Web, Spring Boot DevTools, etc.
3. Click "Generate", unzip, and open in IDE (Eclipse, IntelliJ, VS Code).
4. The main class looks like:

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

@SpringBootApplication is a combination of:

* @Configuration
* @EnableAutoConfiguration
* @ComponentScan

**2. Explain the Need and Benefits of Spring Boot**

Spring Boot is needed to simplify Spring-based Java development and accelerate the development process.

|  |  |
| --- | --- |
| **Feature** | **Benefit** |
| Avoids Boilerplate Code | Uses auto-configuration and annotations |
| No XML Configuration | Uses Java-based configuration |
| Fast Development | Built-in Tomcat/Jetty server, no deployment needed |
| Easy Testing | Embedded servers and starter dependencies ease unit/integration testing |
| Production Ready | Metrics, health checks, and externalized configurations included |

**3. Demonstrate Loading Bean from Spring Configuration File**

In traditional Spring (before Spring Boot), beans were configured via XML.

Step-by-Step Demo:

beans.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="student" class="com.example.Student">

<property name="name" value="Akhila"/>

<property name="department" value="CSE"/>

</bean>

</beans>

Java Class:

public class Student {

private String name;

private String department;

public void display() {

System.out.println("Name: " + name + ", Department: " + department);

}

}

**Application:**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

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

Student s = (Student) context.getBean("student");

s.display();

}

}

**4. Demonstrate Inclusion of Logging in Spring Boot Application**

Spring Boot uses SLF4J and Logback for logging.

Steps to Add Logging:

Add to DemoApplication.java:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@SpringBootApplication

public class DemoApplication {

private static final Logger logger = LoggerFactory.getLogger(DemoApplication.class);

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

logger.info("Application Started");

logger.debug("Debug log activated");

}

}

Customize Logging in application.properties:

server.port=8081

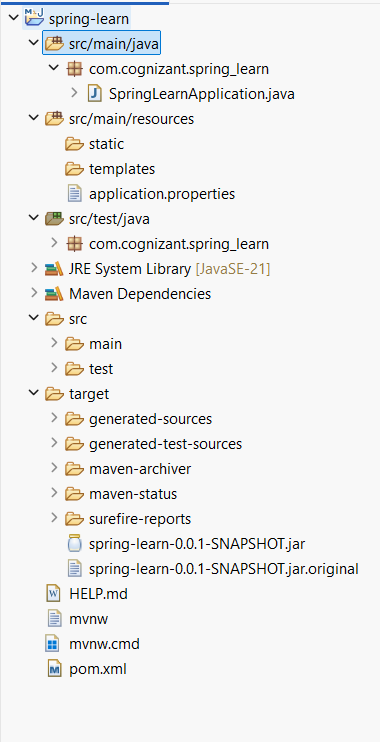
logging.level.root=info

logging.level.com.example=debug

logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %msg%n

**Hands - on 1 Create a Spring Web Project using Maven**

**Dependency Hierarchy**

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-learn</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>21</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**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**

****

**Purpose of @SpringBootApplication**

It serves as the entry point for a Spring Boot application and does the following:

Internally Combines:

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan

**1. @SpringBootConfiguration**

* It is a specialized form of @Configuration.
* It tells Spring that this class contains bean definitions using @Bean methods.

**2. @EnableAutoConfiguration**

* It tells Spring Boot to automatically configure your application based on the dependencies available in the classpath.
* Example: If Spring MVC is in your classpath, it automatically configures a web application.

**3. @ComponentScan**

* It enables component scanning, so Spring will automatically detect and register:
  + @Component
  + @Service
  + @Repository
  + @Controller
  + Other annotated classes in the same package or sub-packages as the main class.

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

**SpringApplication.java**

package com.cognizant.spring\_learn;

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 displayCountry() {

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

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

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

}

public static void main(String[] args) {

*LOGGER*.debug("Start of Main method.");

*displayCountry*();

*LOGGER*.debug("End of Main method.");

}

}

**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.spring\_learn</groupId>

<artifactId>spring-core-country</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

<!-- SLF4J and Logback for Logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.11</version>

</dependency>

</dependencies>

</project>

**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.spring\_learn.Country"*>

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

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

</bean>

</beans>

**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("Inside getCode.");

return code;

}

public void setCode(String code) {

*LOGGER*.debug("Inside setCode.");

this.code = code;

}

public String getName() {

*LOGGER*.debug("Inside getName.");

return name;

}

public void setName(String name) {

*LOGGER*.debug("Inside setName.");

this.name = name;

}

@Override

public String toString() {

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

}

}

**logback.xml**

<configuration>

<appender name=*"STDOUT"* class=*"ch.qos.logback.core.ConsoleAppender"*>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n</pattern>

</encoder>

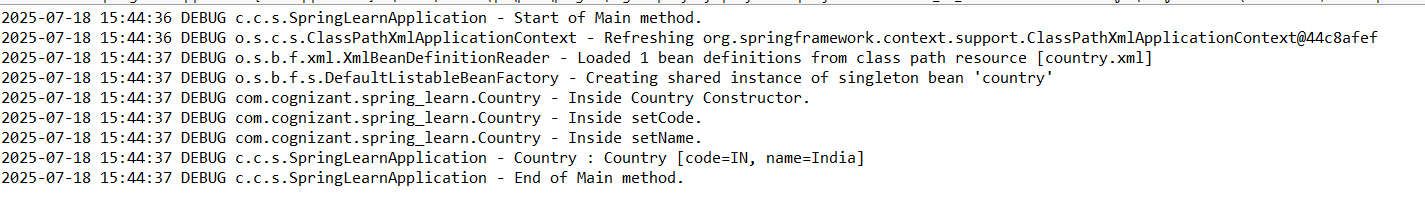
</appender>

<root level=*"debug"*>

<appender-ref ref=*"STDOUT"* />

</root>

</configuration>

**Output**

**1. bean tag, id attribute, class attribute, property tag, name attribute, value attribute**

These are all parts of a Spring XML configuration file (country.xml in your case) used to define and inject dependencies (beans).

**<bean> tag:**

* This is used to declare a bean (object) in the Spring XML configuration file.
* Spring will manage the lifecycle and dependency injection of this bean.

**id attribute:**

* Specifies the name of the bean by which it can be retrieved.
* Example: <bean id="country" ... /> means we can fetch it using context.getBean("country").

**class attribute:**

* Fully qualified name of the class to be instantiated as a bean.
* Example: class="com.cognizant.springlearn.Country"

**<property> tag:**

* Used to set values to the fields of the bean (through setter methods).

name attribute in property:

* Indicates the property/field name of the bean class.

value attribute in property:

* The value to be injected into the property.

**Example:**

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

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

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

</bean>

This will:

* Create an object of Country class
* Set its code field to IN and name to India

**2. ApplicationContext, ClassPathXmlApplicationContext**

These are part of Spring’s Core Container used to load and manage beans.

**ApplicationContext:**

* It is an interface representing the Spring IoC container.
* It is responsible for:
  + Instantiating beans
  + Configuring them
  + Assembling dependencies
* It also supports internationalization, event propagation, and application lifecycle.

**ClassPathXmlApplicationContext:**

* This is a concrete implementation of ApplicationContext.
* It loads context definitions from an XML file present in the classpath.
* Commonly used in standalone Spring applications.

**Example:**

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

This reads country.xml from the classpath and creates the bean(s) defined there.

**3. What happens when context.getBean() is invoked?**

When you call:

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

The following steps occur:

1. Spring looks for a bean with id="country" in the configuration file.
2. Creates an instance of the class specified (com.cognizant.springlearn.Country).
3. Calls the no-argument constructor of that class.
4. Injects values into the fields using the setter methods (setCode(), setName()).
5. Returns the fully initialized object of type Country.

If @Slf4j or proper logging is used, you will see messages like:

Inside Country Constructor.

Setting code to IN

Getting code: IN

**Hello World RESTful Web Service**

**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*.info("Start sayHello()");

String message = "Hello World!!";

*LOGGER*.info("End sayHello()");

return message;

}

}

**application.properties**

spring.application.name=spring-learn

server.port=8083

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

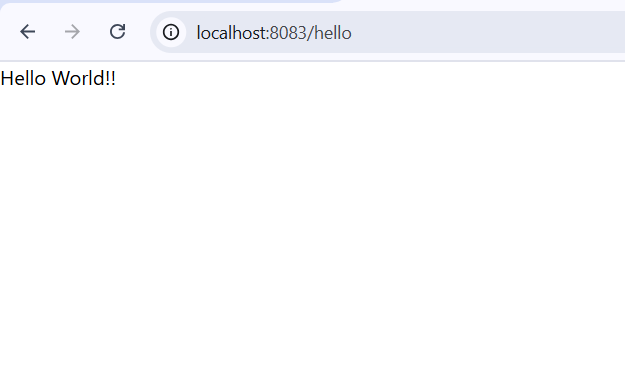
public static void main(String[] args) {

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

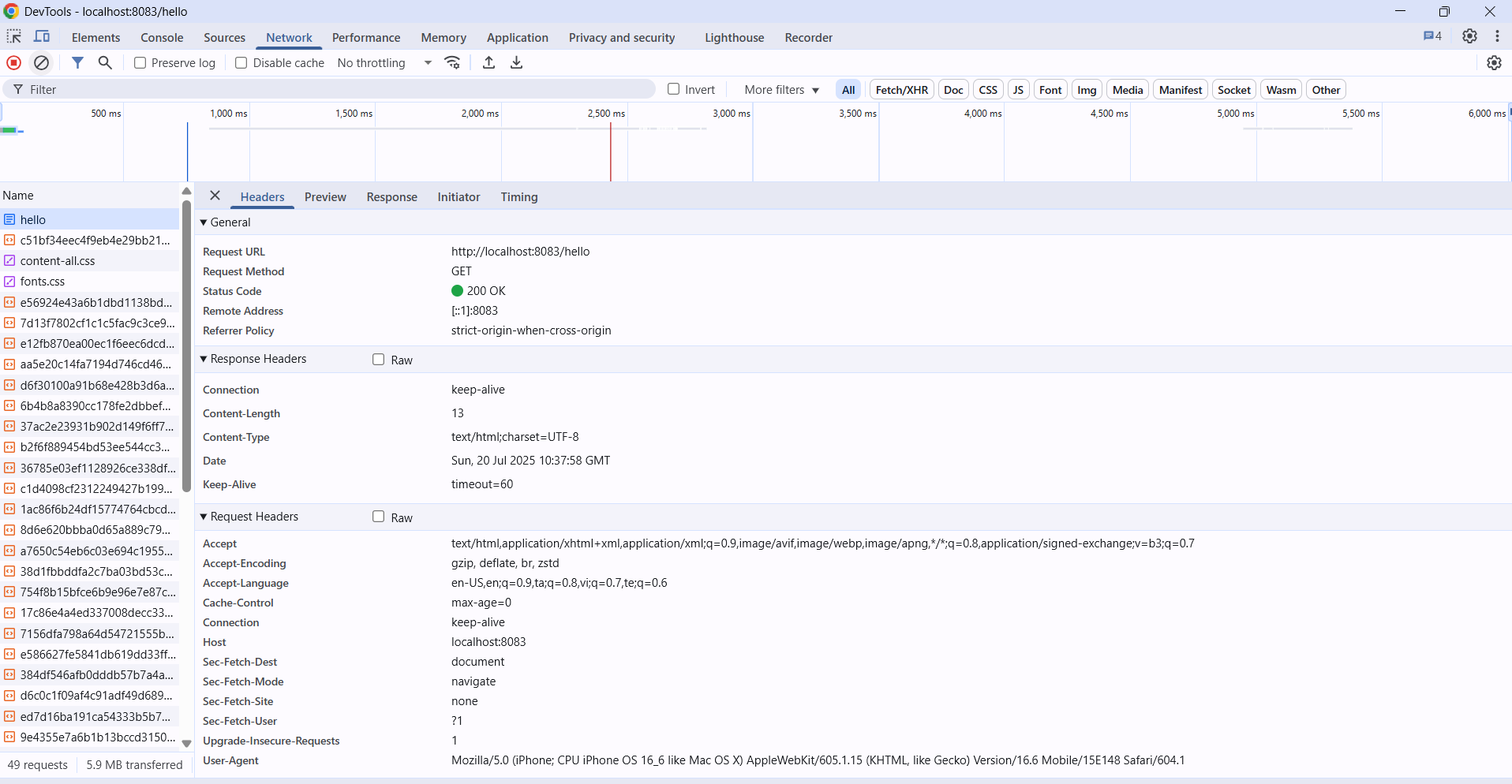
}

}

**Output**

****

**HTTP Header Details In Network Tab**

****

**REST - Country Web Service**

**CountryController.java**

package com.cognizant.spring\_learn.controller;

import com.cognizant.spring\_learn.model.Country;

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

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

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

@RestController

public class CountryController {

@RequestMapping("/country")

public Country getCountryIndia() {

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

Country country = (Country) context.getBean("in");

return country;

}

}

**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;

}

}

**application.properties**

spring.application.name=spring-learn

server.port=8080

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

@ComponentScan(basePackages = "com.cognizant.spring\_learn")

public class SpringLearnApplication {

public static void main(String[] args) {

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

}

}

**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.spring\_learn.model.Country"*>

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

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

</bean>

</beans>

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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>

<description>Spring Learn REST Application</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.18</version> <!-- Or use the version you are targeting -->

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>1.8</java.version> <!-- Or use 17 if your JDK is newer -->

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>2.7.18</version>

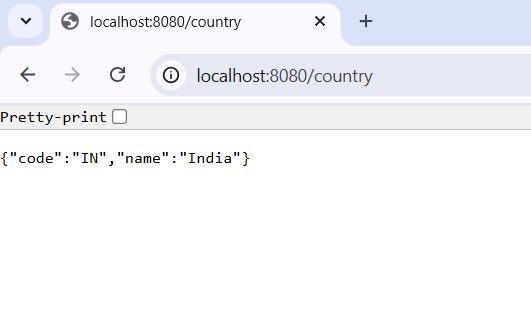
</plugin>

</plugins>

</build>

</project>

**Output**



**SME Explanation**

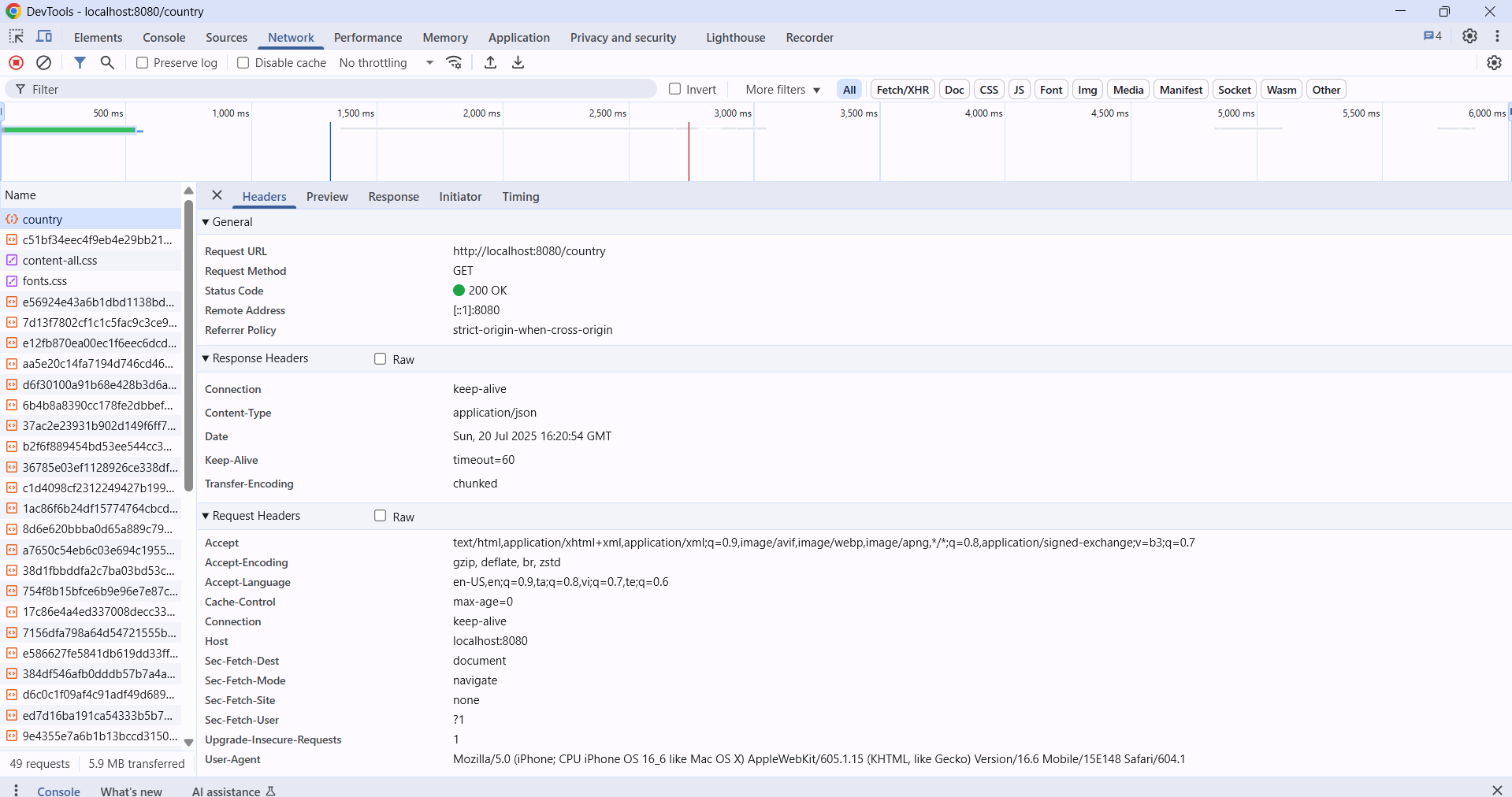
**What happens in the controller method?**

* The method getCountryIndia() is mapped to /country URL.
* It loads ApplicationContext using country.xml.
* Fetches the bean in defined in XML.
* Returns it as a response.
* Spring Boot automatically converts the bean to JSON.

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

* Spring Boot uses **Jackson** (auto-included in spring-boot-starter-web) to convert POJOs to JSON.
* It serializes the Country object by calling getters and building a JSON structure.

**HTTP Header Details in Network Tab**



**REST - Get country based on country code**

**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("/countries/{code}")

public Country getCountry(@PathVariable String code) {

return countryService.getCountry(code);

}

}

**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.toUpperCase();

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**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(country -> country.getCode().equalsIgnoreCase(code))

.findFirst()

.orElse(null);

}

}

**application.properties**

spring.application.name=spring-learn

server.port=8082

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

@ComponentScan(basePackages = "com.cognizant.spring\_learn")

public class SpringLearnApplication {

public static void main(String[] args) {

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

}

}

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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>

<description>Spring Learn REST Application</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.18</version>

<relativePath/>

</parent>

<properties>

<java.version>1.8</java.version> <!-- Use 17 if using JDK 17 -->

</properties>

<dependencies>

<!-- Spring Boot Web Starter -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Spring Boot Maven Plugin -->

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>2.7.18</version>

</plugin>

</plugins>

</build>

</project>

**Output**

**Create authentication service that returns JWT**

**AuthRequest.java**

package com.cognizant.spring\_learn.model;

public class AuthRequest {

private String username;

private String password;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

**SecurityConfig.java**

package com.cognizant.spring\_learn.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import com.cognizant.spring\_learn.service.JwtFilter;

@Configuration

public class SecurityConfig {

@Bean

public UserDetailsService userDetailsService() {

var user = User.withUsername("akhila")

.password(passwordEncoder().encode("12345"))

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authManager(AuthenticationConfiguration config) throws Exception {

return config.getAuthenticationManager();

}

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http, JwtFilter jwtFilter) throws Exception {

http.csrf(csrf -> csrf.disable())

.authorizeHttpRequests(auth -> auth

.requestMatchers("/authenticate").permitAll()

.anyRequest().authenticated())

.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

return http.build();

}

}

**JwtResponse.java**

package com.cognizant.spring\_learn.model;

public class JwtResponse {

private String token;

public JwtResponse(String token) {

this.token = token;

}

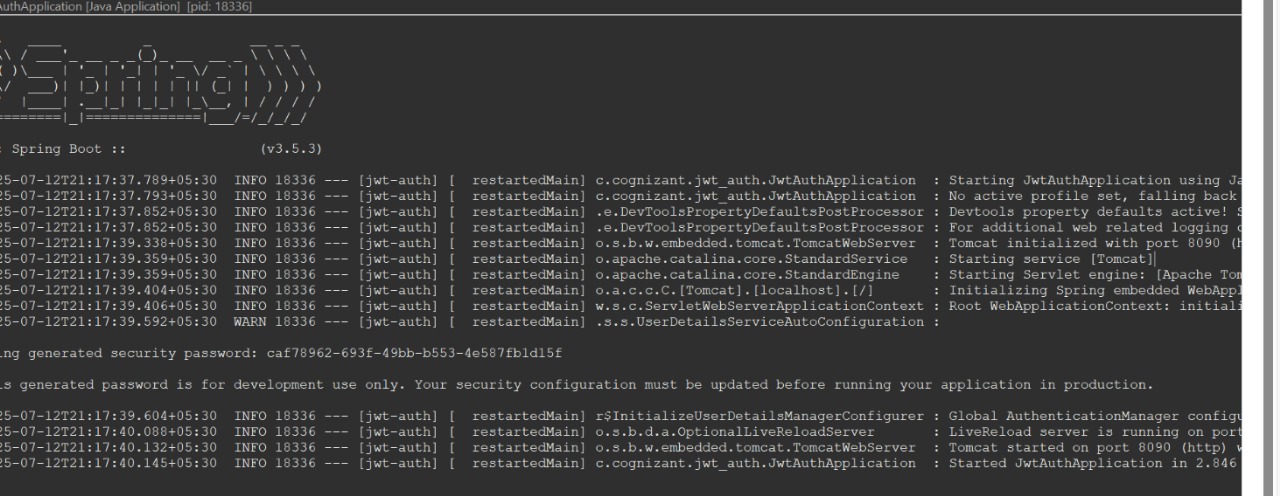
public String getToken() {

return token;

}

}

**Output**

****