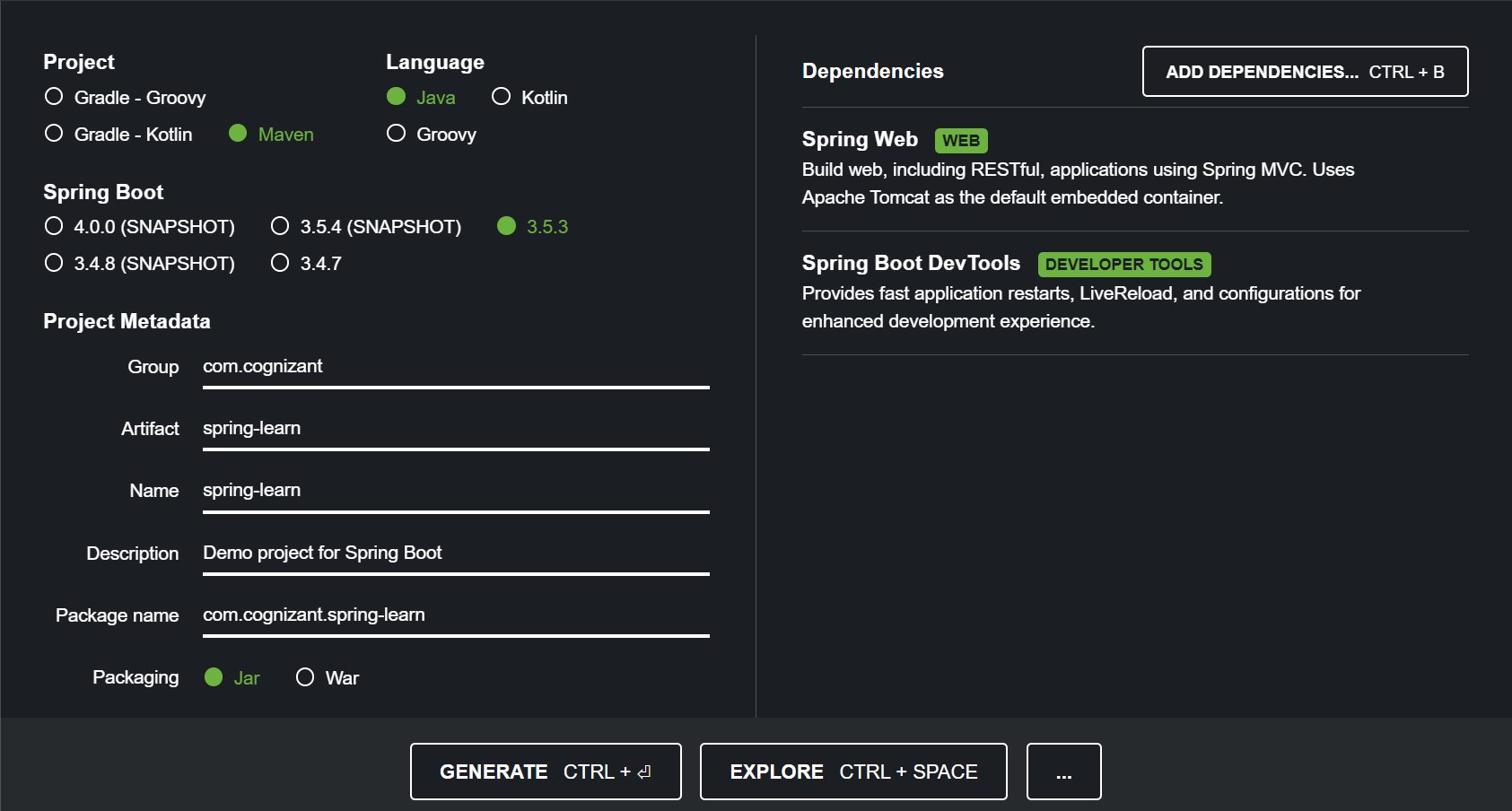
**Hands on 1**

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

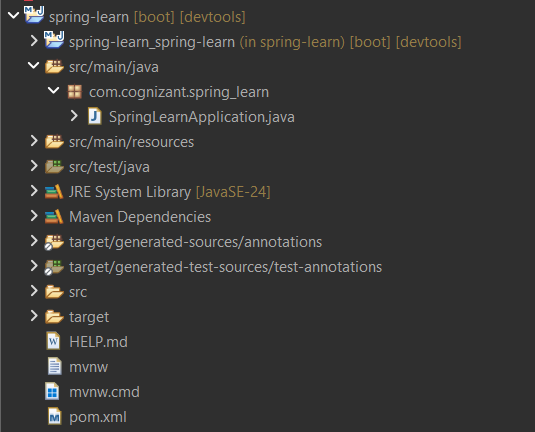
1. Go to <https://start.spring.io/>
2. Change Group as “com.cognizant”
3. Change Artifact Id as “spring-learn”
4. Select Spring Boot DevTools and Spring Web



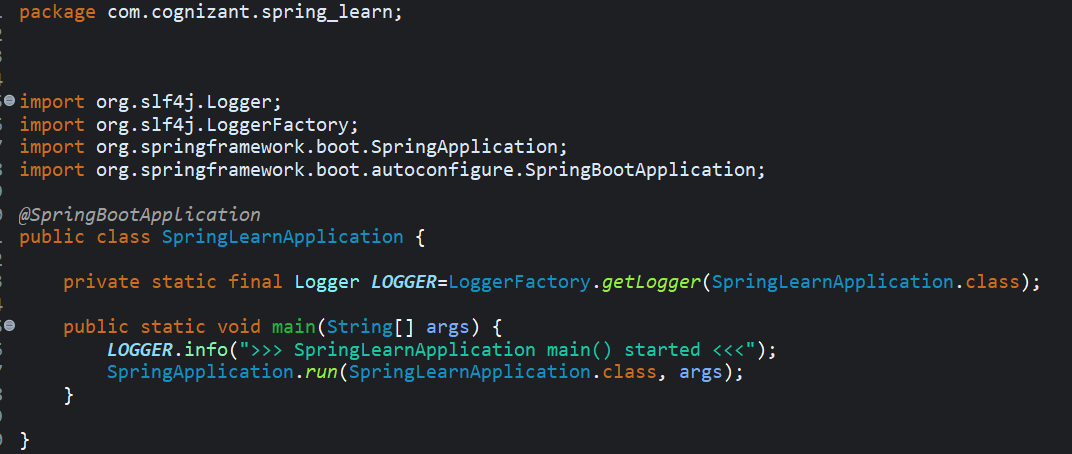
1. Create and download the project as zip
2. Extract the zip in root folder to Eclipse Workspace
3. Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
4. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
5. Include logs to verify if main() method of SpringLearnApplication.
6. Run the SpringLearnApplication class.

SME to walk through the following aspects related to the project created:

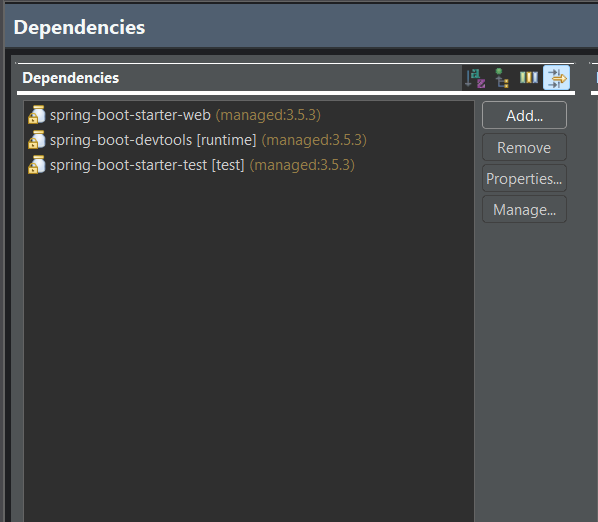
1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application



1. SpringLearnApplication.java - Walkthrough the main() method.

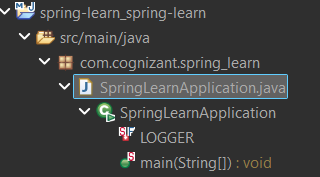


1. Purpose of @SpringBootApplication annotation
2. pom.xml
   1. Walkthrough all the configuration defined in XML file
3. <?xml version="1.0" encoding="UTF-8"?>
4. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
6. <modelVersion>4.0.0</modelVersion>
7. <parent>
8. <groupId>org.springframework.boot</groupId>
9. <artifactId>spring-boot-starter-parent</artifactId>
10. <version>3.5.3</version>
11. <relativePath/> <!-- lookup parent from repository -->
12. </parent>
13. <groupId>com.cognizant</groupId>
14. <artifactId>spring-learn</artifactId>
15. <version>0.0.1-SNAPSHOT</version>
16. <name>spring-learn</name>
17. <description>Demo project for Spring Boot</description>
18. <url/>
19. <licenses>
20. <license/>
21. </licenses>
22. <developers>
23. <developer/>
24. </developers>
25. <scm>
26. <connection/>
27. <developerConnection/>
28. <tag/>
29. <url/>
30. </scm>
31. <properties>
32. <java.version>24</java.version>
33. </properties>
34. <dependencies>
35. <dependency>
36. <groupId>org.springframework.boot</groupId>
37. <artifactId>spring-boot-starter-web</artifactId>
38. </dependency>
39. <dependency>
40. <groupId>org.springframework.boot</groupId>
41. <artifactId>spring-boot-devtools</artifactId>
42. <scope>runtime</scope>
43. <optional>true</optional>
44. </dependency>
45. <dependency>
46. <groupId>org.springframework.boot</groupId>
47. <artifactId>spring-boot-starter-test</artifactId>
48. <scope>test</scope>
49. </dependency>
50. </dependencies>
51. <build>
52. <plugins>
53. <plugin>
54. <groupId>org.springframework.boot</groupId>
55. <artifactId>spring-boot-maven-plugin</artifactId>
56. </plugin>
57. </plugins>
58. </build>
59. </project>
    1. Open 'Dependency Hierarchy' and show the dependency tree.



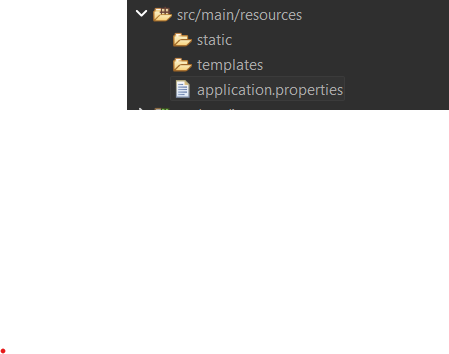
SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code



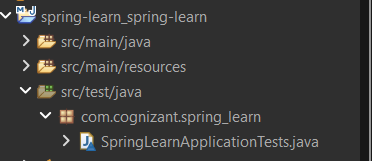
1. src/main/resources - Folder for application configuration

This folder is used for Spring Boot configuration files. The application.properties file is where we set up server port, logging, DB config, etc

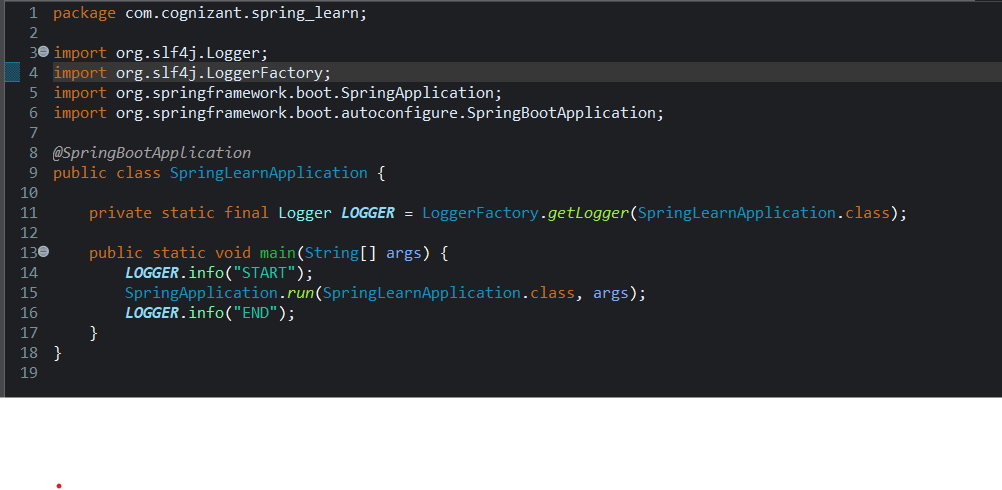


1. src/test/java - Folder with code for testing the application

This folder is for writing test cases to validate the business logic. Spring Boot uses JUnit by default for testing.



1. SpringLearnApplication.java - Walkthrough the main() method.

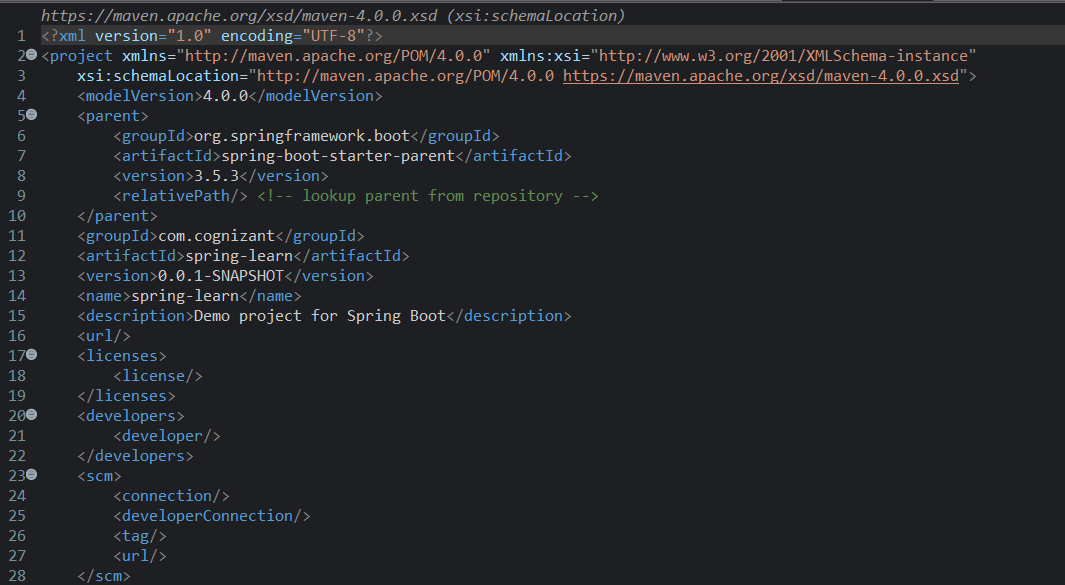


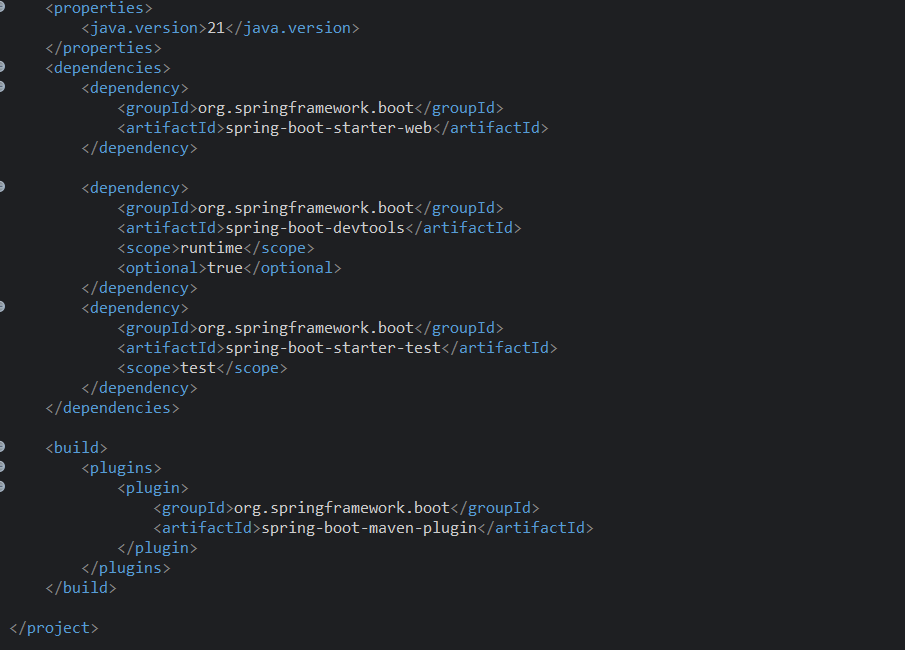
1. Purpose of @SpringBootApplication annotation

This annotation is used for:

@Configuration : Marks class as a configuration source. @EnableAutoConfiguration : Enables Spring Boot’s auto configuration. @ComponentScan : Scans the package for components, services, etc.

1. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.





**Hands on 4**

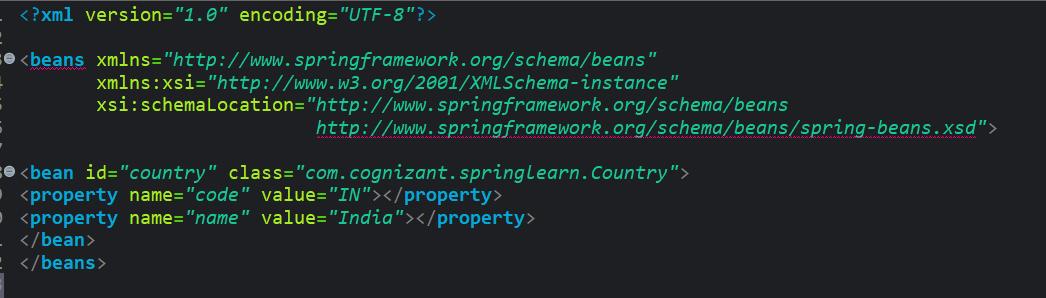
**Spring Core – Load Country from Spring Configuration XML**   
  
An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country. 

|  |  |
| --- | --- |
| **Code** | **Name** |
| US | United States |
| DE | Germany |
| IN | India |
| JP | Japan |

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details.  
  
Steps to implement

* Pick any one of your choice country to configure in Spring XML configuration named country.xml.
* Create a bean tag in spring configuration for country and set the property and values

country.xml



* Create Country class with following aspects:
  + Instance variables for code and name
  + Implement empty parameter constructor with inclusion of debug log within the constructor with log message as “Inside Country Constructor.”
  + Generate getters and setters with inclusion of debug with relevant message within each setter and getter method.
  + Generate toString() method
* package com.cognizant.springlearn;
* import org.slf4j.Logger;
* import org.slf4j.LoggerFactory;
* public class Country {
* Logger LOGGER = LoggerFactory.*getLogger*(Country.class);
* private String name;
* private String code;

* public String getName() {
* LOGGER.debug("Inside getName");
* return name;
* }
* public void setName(String name) {
* LOGGER.debug("Inside setName");
* this.name = name;
* }
* public String getCode() {
* LOGGER.debug("Inside getCode");
* return code;
* }
* public void setCode(String code) {
* LOGGER.debug("Inside setCode");
* this.code = code;
* }
* public Country() {
* LOGGER.debug("Inside Country Constructor");
* }
* *@Override*
* public String toString() {
* return "Country [name=" + name + ", code=" + code + "]";
* }
* }
* Create a method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country details. ClassPathXmlApplicationContext, ApplicationContext and context.getBean(“beanId”, Country.class). Refer sample code for displayCountry() method below.

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

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

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

* Invoke displayCountry() method in main() method of SpringLearnApplication.java.
* Execute main() method and check the logs to find out which constructors and methods were invoked.
* package com.cognizant.springlearn;
* import org.slf4j.Logger;
* import org.slf4j.LoggerFactory;
* import org.springframework.boot.SpringApplication;
* import org.springframework.boot.autoconfigure.SpringBootApplication;
* import org.springframework.context.ApplicationContext;
* import org.springframework.context.support.ClassPathXmlApplicationContext;
* *@SpringBootApplication*
* public class SpringLearnApplication {
* private static final Logger ***LOGGER***=LoggerFactory.*getLogger*(SpringLearnApplication.class);

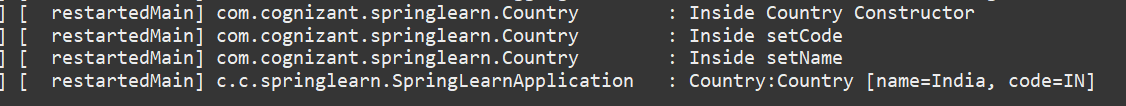
* public static void displayCountry() {
* ApplicationContext context= new ClassPathXmlApplicationContext("country.xml");
* Country country=context.getBean("country",Country.class);
* ***LOGGER***.debug("Country:{}",country.toString());
* }
* public static void main(String[] args) {
* ***LOGGER***.info(">>> SpringLearnApplication main() started <<<");
* SpringApplication.*run*(SpringLearnApplication.class, args);
* *displayCountry*();
* }
* }

**application.properties**

spring.application.name=spring-learn

logging.level.com.cognizant.springlearn=DEBUG

**Output**

****

SME to provide more detailing about the following aspects:

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

<bean>: Declares a Spring bean.(id:) Unique name to reference this bean ("country"). (class:) Fully qualified Java class to instantiate. <property>: Used to set values of fields in the bean via setter injection. (name:) Matches Java property name (e.g., code, name).(value:) Value to assign (e.g., IN, India).

* ApplicationContext, ClassPathXmlApplicationContext

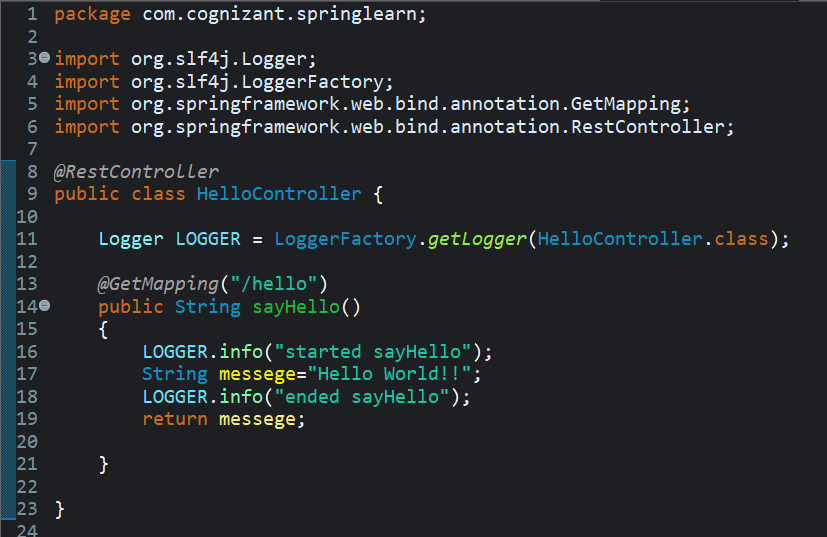
ApplicationContext-it is the core spring container interface to access beans

ClassPathXmlApplicationContext-it loads bean definations from Xml file on the classpath

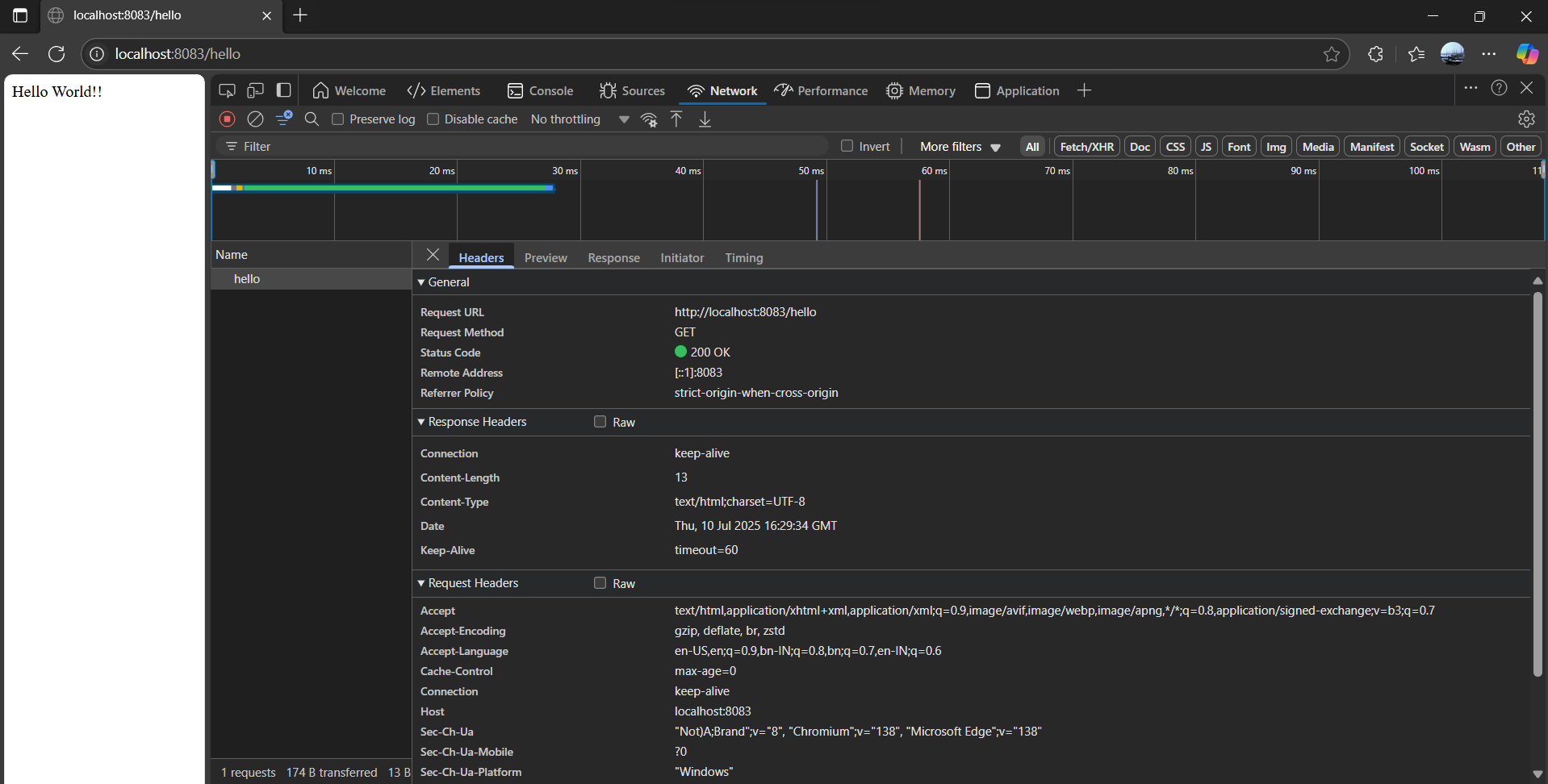
* What exactly happens when context.getBean() is invoked

When context.getBean() in invoked it reads the bean definition from country.xml, sets its properties via setters,return the bean instance cast to country

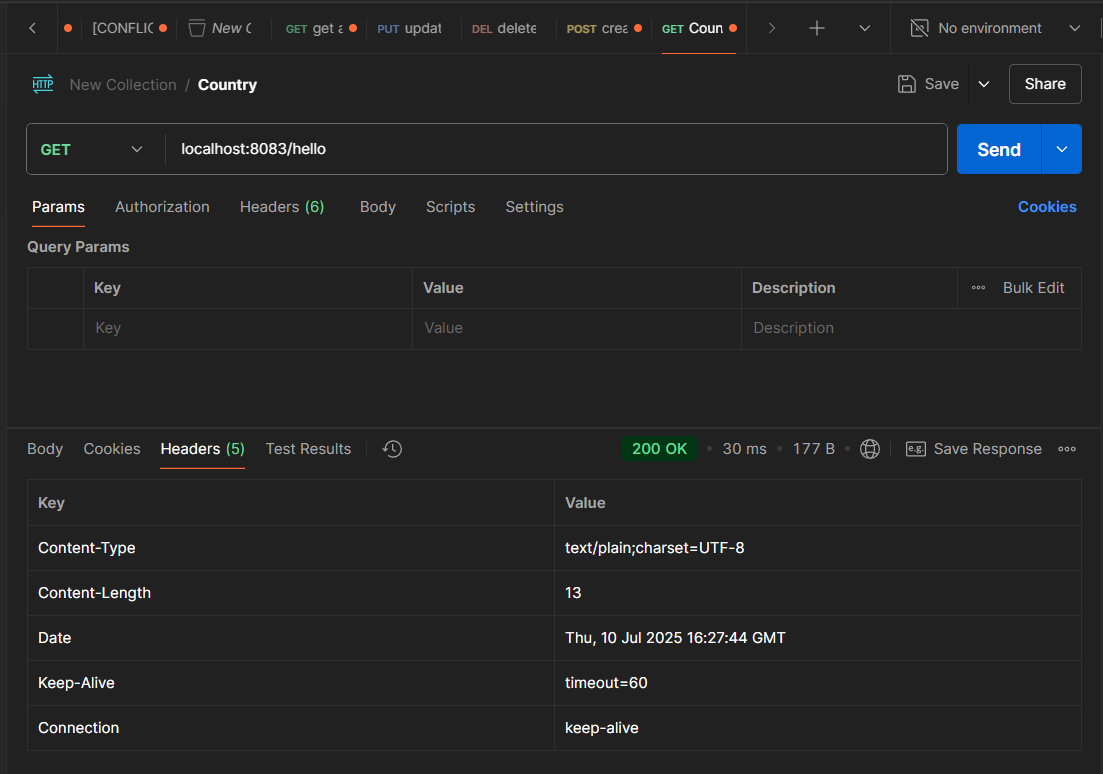
**Hello World RESTful Web Service**   
  
Write a REST service in the spring learn application created earlier, that returns the text "Hello World!!" using Spring Web Framework. Refer details below:  
  
**Method:** GET  
**URL:** /hello  
**Controller:** com.cognizant.spring-learn.controller.HelloController  
**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



**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"

}

**CountryController Class**

package com.cognizant.springlearn.controller;

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;

import com.cognizant.springlearn.Country;

*@RestController*

public class CountryController {

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

*@RequestMapping*("/country")

public Country getCountryIndia() {

LOGGER.info("START getCountryIndia");

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

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

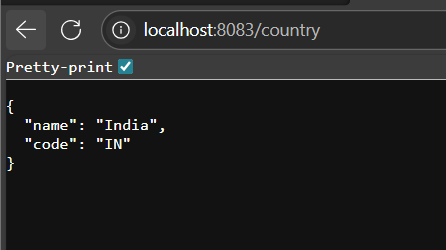
LOGGER.info("END getCountryIndia");

return country;

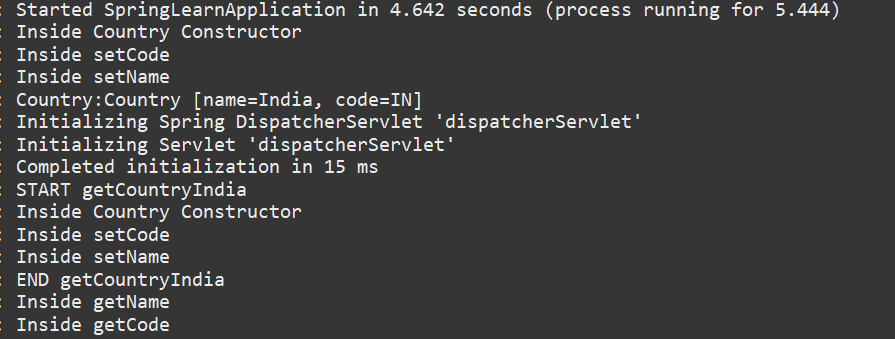
}

}

**Output in browser**

****

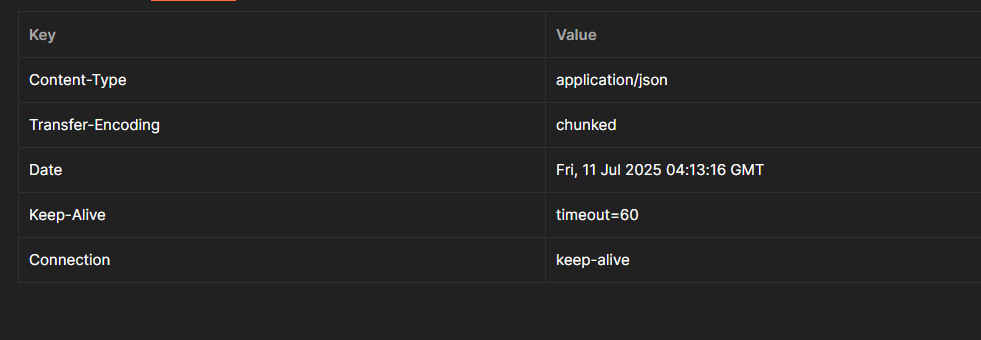
**Console logs**

****

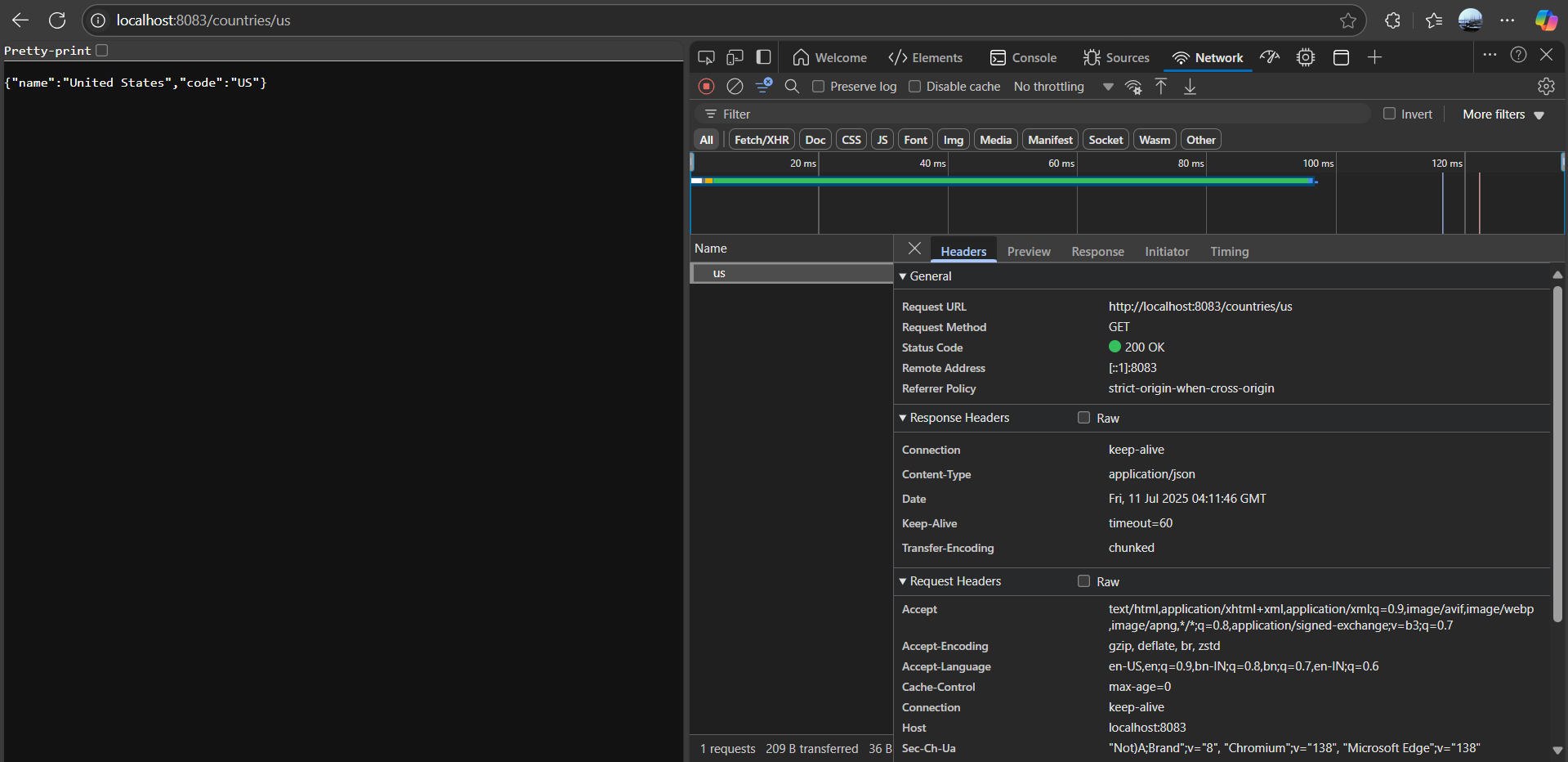
SME to explain the following aspects:

* 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

In postman



In browser



SME to explain the following aspects:

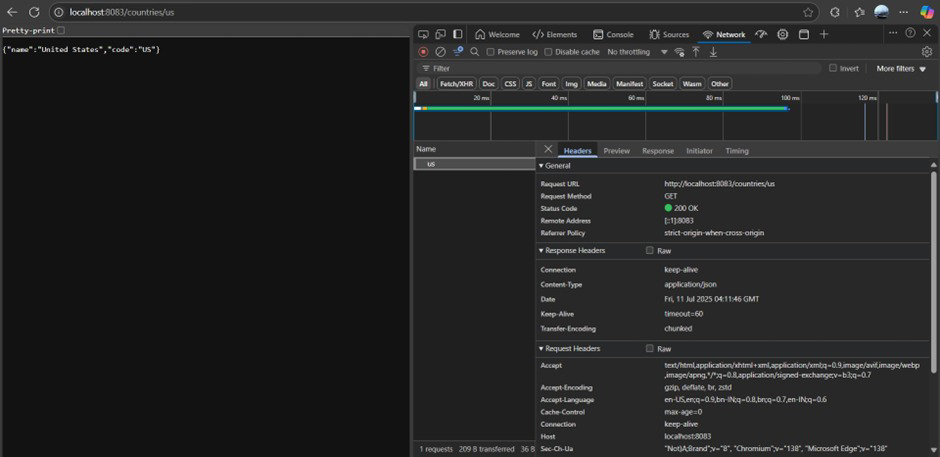
* What happens in the controller method?

getCountryIndia() is mapped to /country using @RequestMapping,it loads the bean country from country.xml file using ApplicationContext.

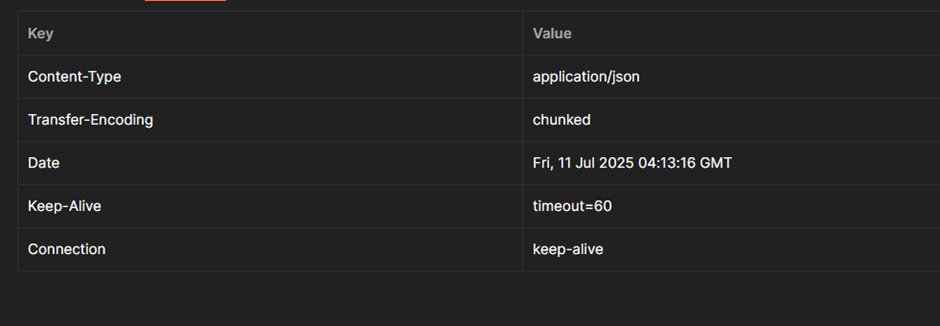
* How the bean is converted into JSON reponse?

SpringBoot uses JSON automatically,in which @RestController implies @ResponseBody where spring automatically converts any POJO into JSON.

* 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



**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)

**CountryController Class**

package com.cognizant.springlearn.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

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

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

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

import com.cognizant.springlearn.Country;

import com.cognizant.springlearn.service.CountryService;

*@RestController*

public class CountryController {

*@Autowired*

private CountryService countryService;

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

*@RequestMapping*("/country")

public Country getCountryIndia() {

***LOGGER***.info("START getCountryIndia");

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

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

***LOGGER***.info("END getCountryIndia");

return country;

}

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

public Country getCountry(*@PathVariable* String code) {

***LOGGER***.info("START getCountry() with code:{}",code);

Country country=countryService.getCountry(code);

***LOGGER***.info("END getCountry()");

return country;

}

}

**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

**CountryService Class**

package com.cognizant.springlearn.service;

import java.util.List;

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 com.cognizant.springlearn.Country;

*@Service*

public class CountryService {

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

public Country getCountry(String code)

{

***LOGGER***.debug("START getCountry()");

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

List<Country> countries=(List<Country>) context.getBean("countryList");

Country result=countries.stream().filter(c->c.getCode().equalsIgnoreCase(code)).findFirst().orElse(null);

***LOGGER***.debug("END getCountry()");

return result;

}

}

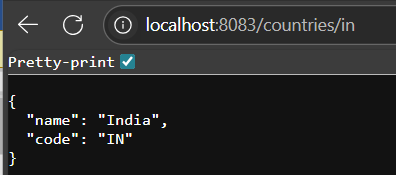
**Sample Request**: http://localhost:8083/country/in  
  
**Sample Response**:

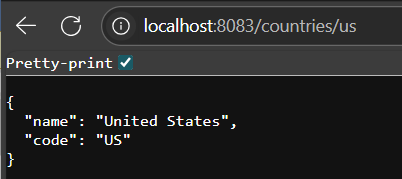
{

  "code": "IN",

  "name": "India"

}





**JWT-handson**

**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.

**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 that sets Spring Boot version -->

<parent>

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

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

<version>3.2.4</version> <!-- You can update to 3.5.3 if needed -->

<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>Spring Boot JWT Project</description>

<properties>

<java.version>17</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-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.0</version>

</dependency>

<!-- Fix for Java 11+ compatibility -->

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.1</version>

</dependency>

<!-- For JUnit and Spring Boot testing -->

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Spring Boot Plugin -->

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**SecurityConfig class**

package com.cognizant.springlearn.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

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

import org.springframework.security.web.SecurityFilterChain;

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

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

*@Configuration*

public class SecurityConfig {

*@Bean*

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeHttpRequests()

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

.anyRequest().authenticated()

.and()

.httpBasic();

return http.build();

}

*@Bean*

public AuthenticationManager authManager(HttpSecurity http) throws Exception {

return http.getSharedObject(AuthenticationManagerBuilder.class)

.inMemoryAuthentication()

.withUser("user").password(passwordEncoder().encode("pwd")).roles("USER")

.and()

.withUser("admin").password(passwordEncoder().encode("pwd")).roles("ADMIN")

.and()

.passwordEncoder(passwordEncoder())

.and()

.build();

}

*@Bean*

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

**AuthenticationController class**

package com.cognizant.springlearn.controller;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

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

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

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

import java.util.Base64;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

*@*RestController

public class AuthenticationController {

*@*GetMapping("/authenticate")

public Map<String, String> authenticate(*@*RequestHeader("Authorization") String authHeader) {

System.***out***.println(">>> Inside /authenticate endpoint");

System.***out***.println("Authorization Header: " + authHeader);

String username = getUser(authHeader);

String token = generateJwt(username);

Map<String, String> map = new HashMap<>();

map.put("token", token);

System.***out***.println("Generated Token: " + token);

return map;

}

private String getUser(String authHeader) {

String base64Credentials = authHeader.substring("Basic ".length());

byte[] decoded = Base64.*getDecoder*().decode(base64Credentials);

String decodedStr = new String(decoded);

return decodedStr.split(":")[0]; // returns "user" from "user:pwd"

}

private String generateJwt(String username) {

return Jwts.*builder*()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.*currentTimeMillis*() + 20 \* 60 \* 1000)) // 20 mins

.signWith(*SignatureAlgorithm*.***HS256***, "secretkey")

.compact();

}

}

**SpringLearnApplication Class**

package com.cognizant.springlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

*@SpringBootApplication*

public class SpringLearnApplication {

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

public static void displayCountry() {

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

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

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

}

public static void main(String[] args) {

***LOGGER***.info(">>> SpringLearnApplication main() started <<<");

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

*displayCountry*();

}

}

**Output**

