

WEEK – 4

SPRING-REST-HANDSON

Hands on 1

Create a Spring Web Project using Maven :

Follow steps below to create a project:

Go to <https://start.spring.io/>

- Change Group as “com.cognizant”
- Change Artifact Id as “spring-learn”
- Select Spring Boot DevTools and Spring Web
- Create and download the project as zip
- Extract the zip in root folder to Eclipse Workspace
- 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
- Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
- Include logs to verify if main() method of SpringLearnApplication.

Run the SpringLearnApplication class.

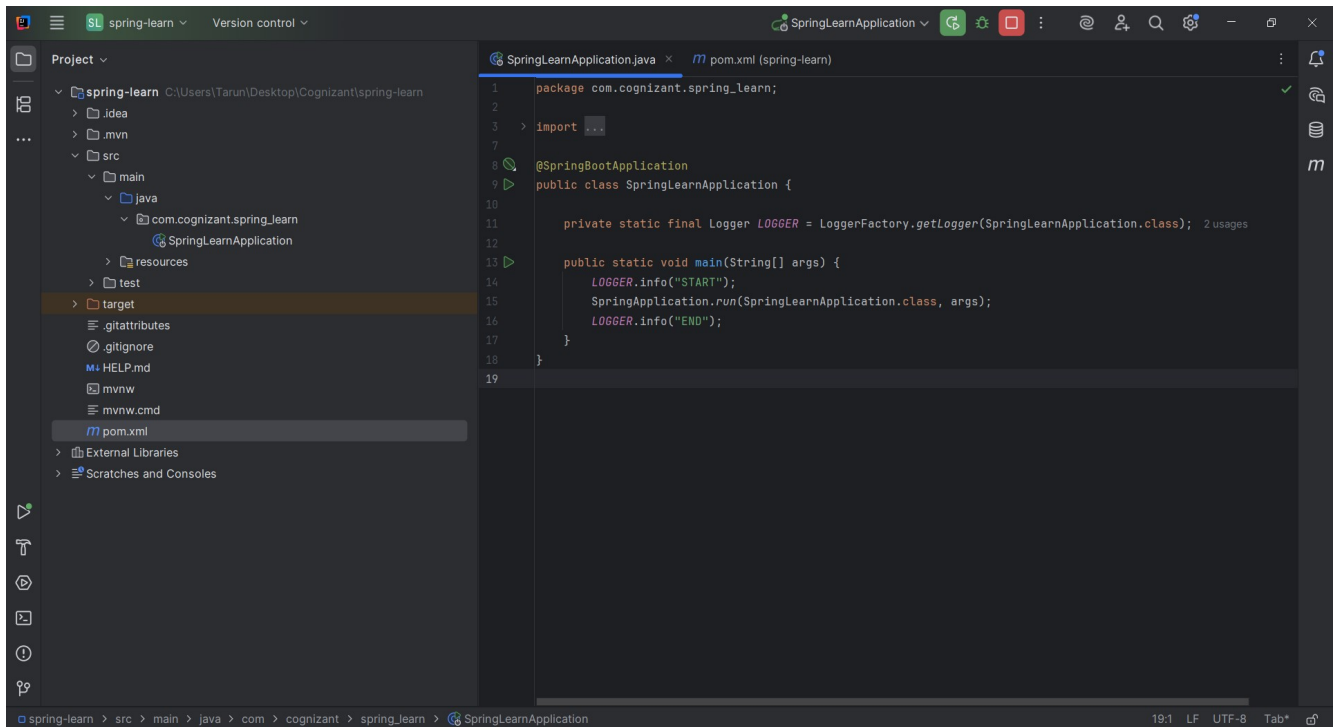
The screenshot shows the Spring Initializr web application interface. The header includes the Spring logo and the text "spring initializr". The main content area is divided into several sections:

- Project:** Radio buttons for "Gradle - Groovy", "Gradle - Kotlin", and "Maven" (selected).
- Language:** Radio buttons for "Java" (selected), "Kotlin", and "Groovy".
- Spring Boot:** Radio buttons for "4.0.0 (SNAPSHOT)", "3.5.4 (SNAPSHOT)", "3.5.3" (selected), "3.4.8 (SNAPSHOT)", and "3.4.7".
- Project Metadata:** Text input fields for "Group" (com.cognizant), "Artifact" (spring-learn), "Name" (spring-learn), "Description" (Demo project for Spring Boot), and "Package name" (com.cognizant.spring-learn).
- Dependencies:** A section with a button "ADD DEPENDENCIES... CTRL + B". It lists "Spring Web" (WEB) and "Spring Boot DevTools" (DEVELOPER TOOLS) with their descriptions.

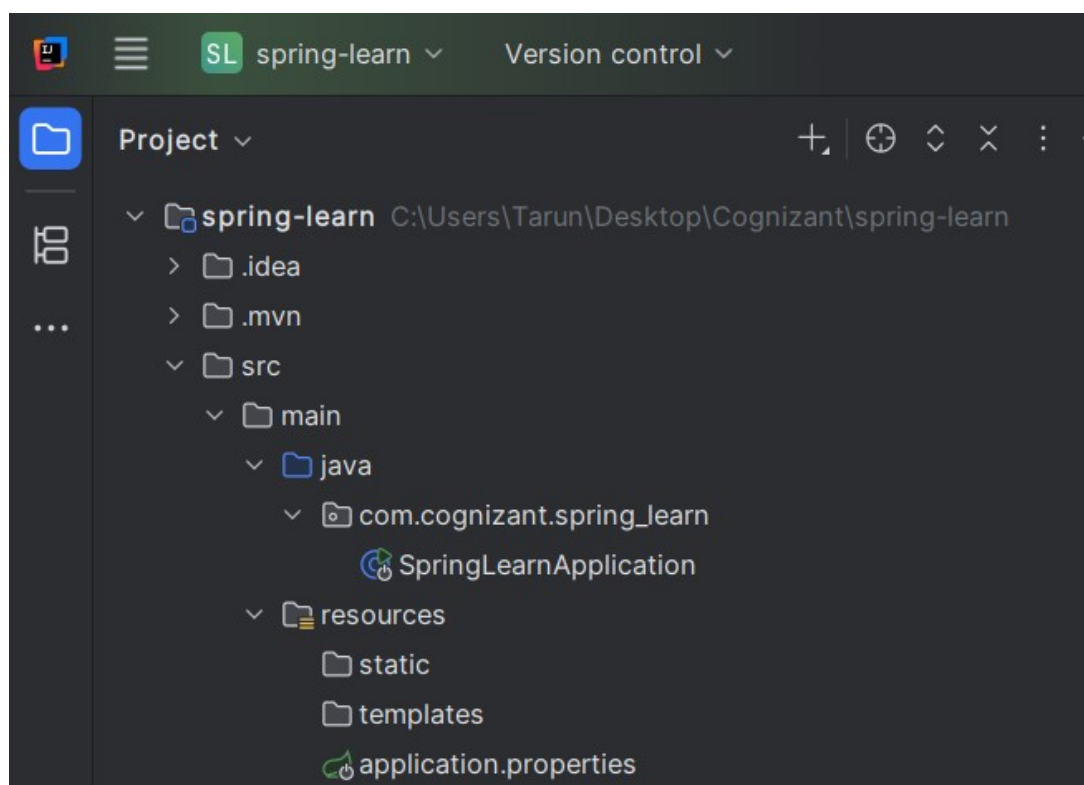
At the bottom, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "...".

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

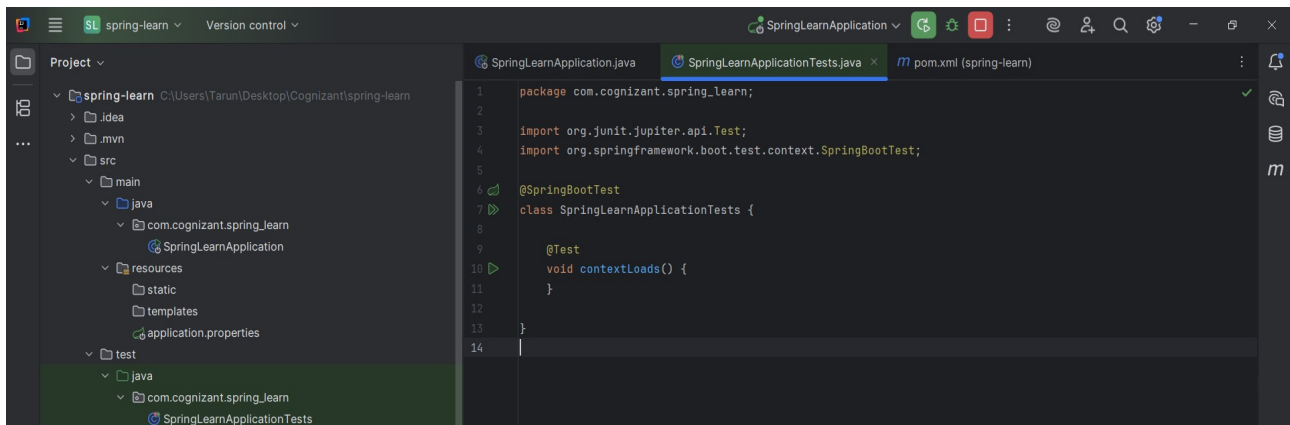
1. src/main/java - Folder with application code :



2. src/main/resources - Folder for application configuration :



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

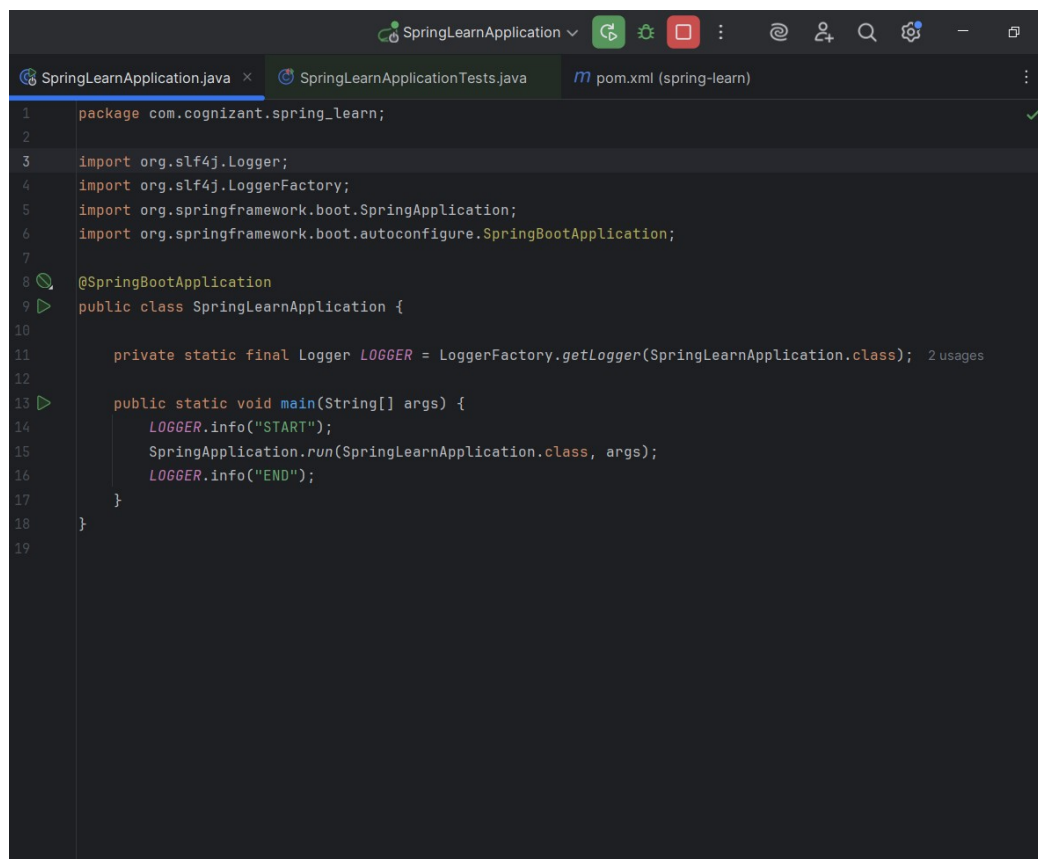


4.SpringLearnApplication.java - Walkthrough the main() method :

- Purpose of @SpringBootApplication annotation :

The **@SpringBootApplication** annotation is used to mark the main class of a Spring Boot application. It is a combination of three important annotations:

- @Configuration – Tells Spring that this class has configuration settings.
- @EnableAutoConfiguration – Automatically configures Spring Boot based on the dependencies.
- @ComponentScan – Scans for components (like services, repositories) in the package and sub-packages.



- Walkthrough all the configuration defined in XML file :

pom.xml :

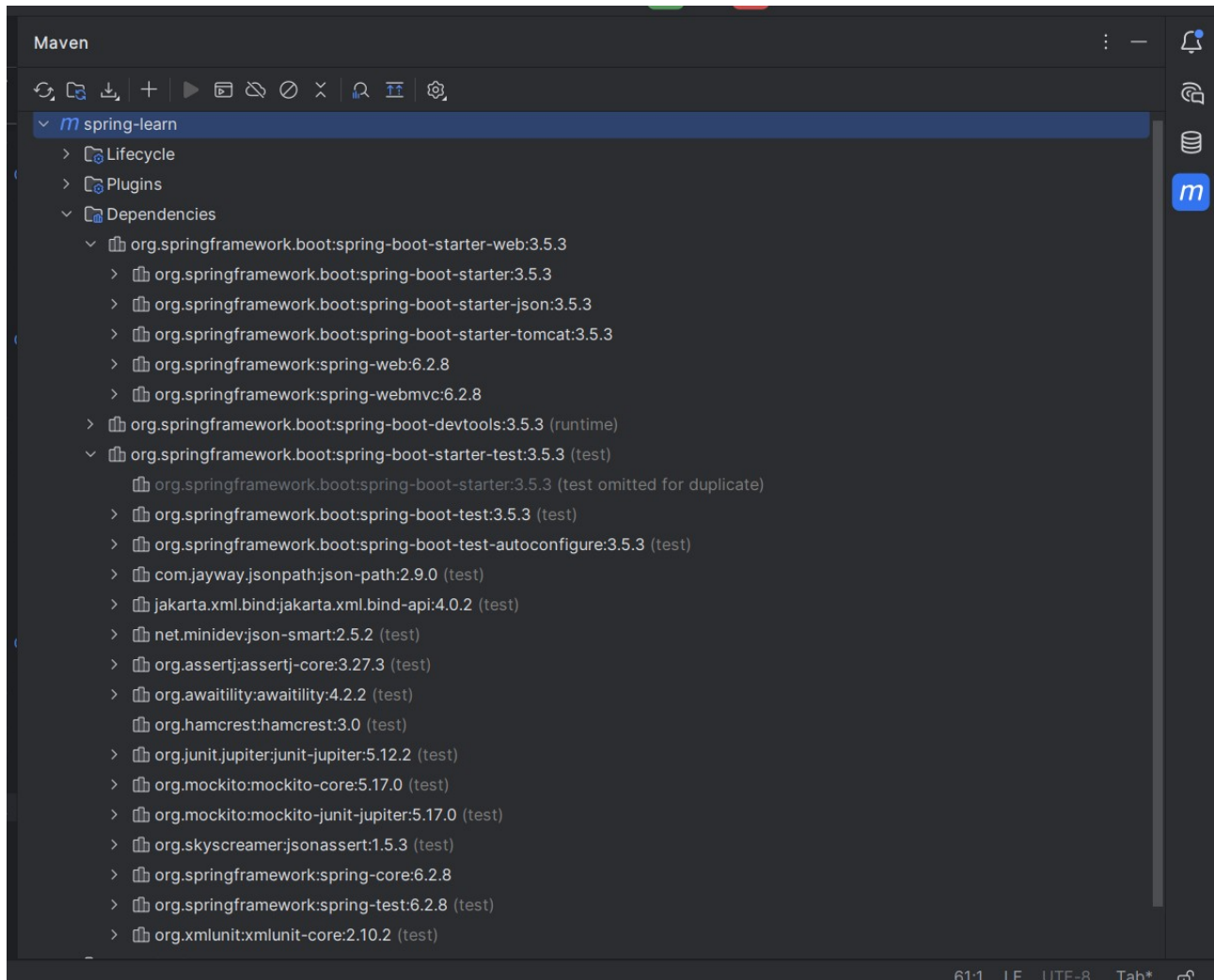
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <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>
```

```
</project>
```

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

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

SME to provide more detailing about the following aspects:

- bean tag, id attribute, class attribute, property tag, name attribute, value attribute
- ApplicationContext, ClassPathXmlApplicationContext
- What exactly happens when context.getBean() is invoked

bean tag :

- It is used in XML to define a Spring-managed object (called a bean).
- This tells Spring which class to instantiate and manage.

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

id attribute :

- Unique name for the bean inside the Spring container.
- Used to retrieve the bean using `getBean("id")`

Example:

id="country" → can be accessed with `context.getBean("country")`.

class attribute :

- Full qualified class name of the bean.
- Tells Spring **which class to instantiate**.

Example:

```
class="com.cognizant.spring_learn.Country"
```

property tag :

- Used to set the values of fields (setters) in the bean class.
- Should match the set method name

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

name attribute (inside property tag) :

- Matches the name of the property (i.e., field name) in the class.
- If the class has setName(String name), then name="name"

value attribute :

- The actual value to be injected into the property.
- Passed to the corresponding setter method.

Example :

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

↑ This creates a Country object, and calls setCode("IN") and setName("India")

ApplicationContext :

- Responsible for instantiating, configuring, and managing beans.
- More advanced than BeanFactory.

ClassPathXmlApplicationContext

- One implementation of ApplicationContext.
- Loads bean definitions from an XML file present in the classpath.

Example :

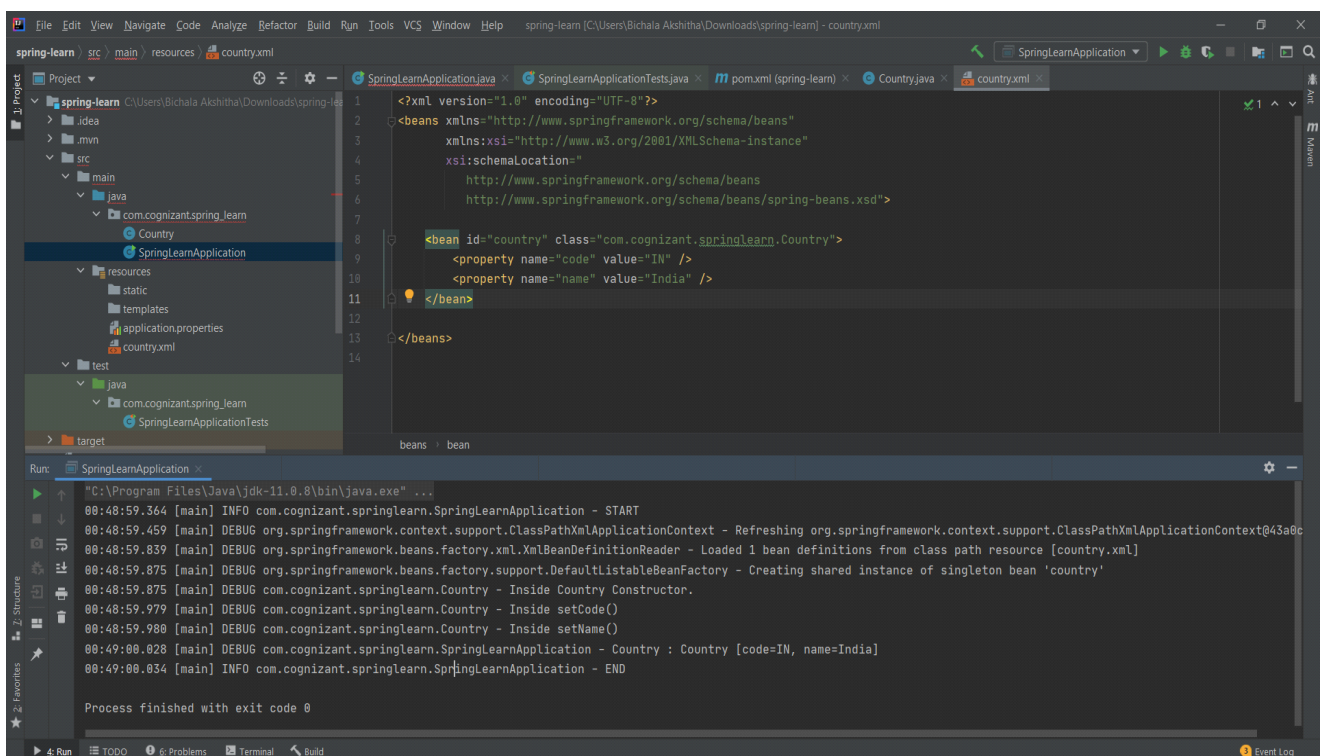
```
ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
```

What happens when context.getBean() is invoked?

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

1. Spring searches for the bean with ID "country" in the XML.
2. It checks the class: com.cognizant.spring_learn.Country.
3. Creates an **object of that class**.
4. Sets the values using setters (based on <property> tags).
5. Returns the **fully initialized bean**.

OUTPUT :



The screenshot displays an IDE with the following components:

- Project Explorer:** Shows the project structure with folders like `src/main/resources` and `src/test/resources`. The `country.xml` file is highlighted in the `resources` folder.
- Editor:** Displays the content of `country.xml`, which defines a bean named "country" of type `com.cognizant.spring_learn.Country` with properties `code="IN"` and `name="India"`.
- Run Console:** Shows the output of the application. The log indicates that the Spring context is refreshed, the bean definitions are loaded, and the bean "country" is created and initialized with the specified properties. The application ends with an exit code of 0.

Hello World RESTful Service using Spring Web Framework :

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

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);
    }
}
```

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;
    }
}
```

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>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.5.3</version>
        <relativePath/>
```

```
</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>

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

<dependencies>
    <!-- Core Web Starter -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Devtools for hot reload -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>

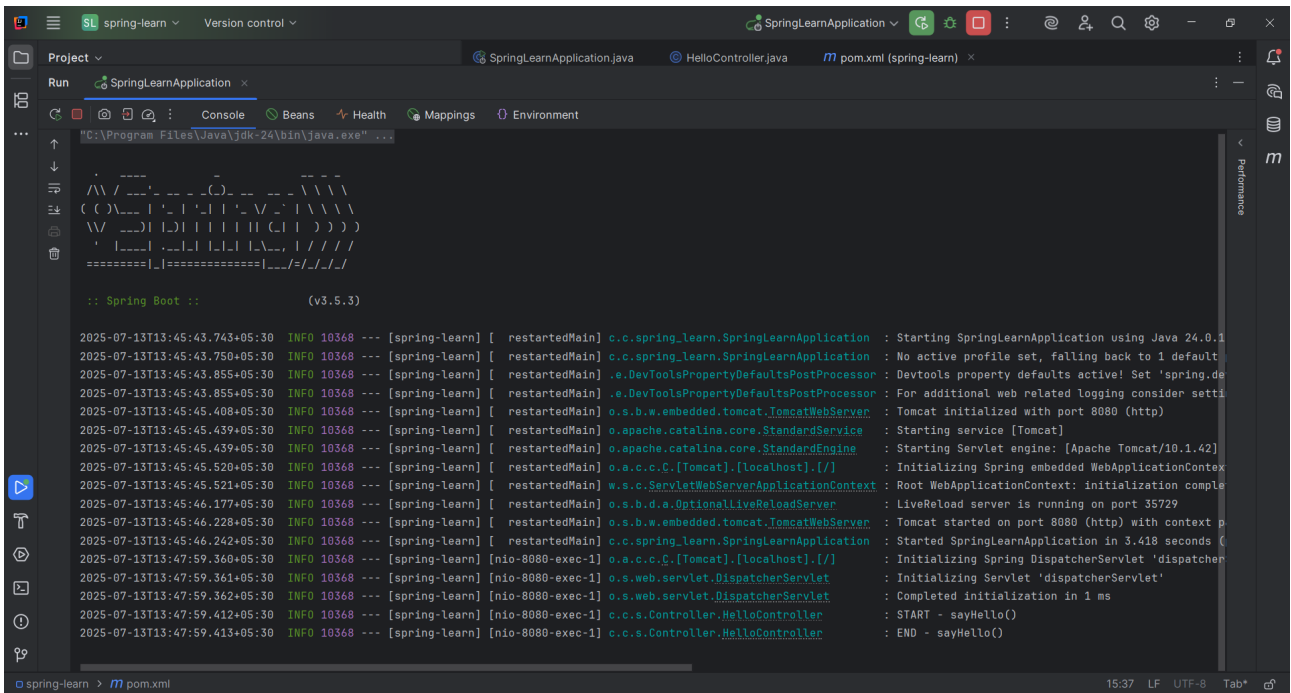
    <!-- Unit Testing -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- Spring Context (for XML config loading) -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>

    <!-- Logging API (already handled by spring-boot-starter, but you can keep this) -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

OUTPUT :



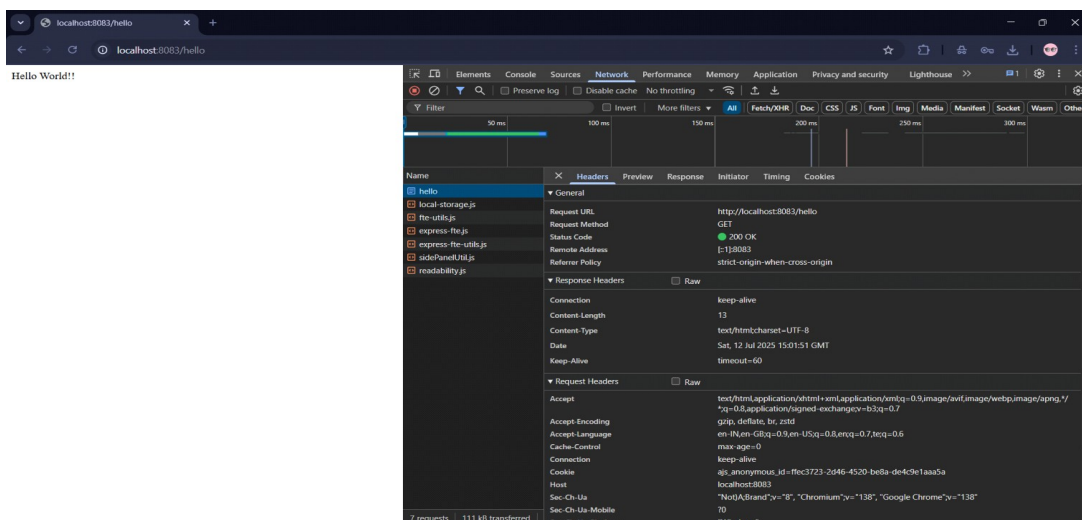
Try the URL <http://localhost:8083/hello> in both chrome browser and postman. :

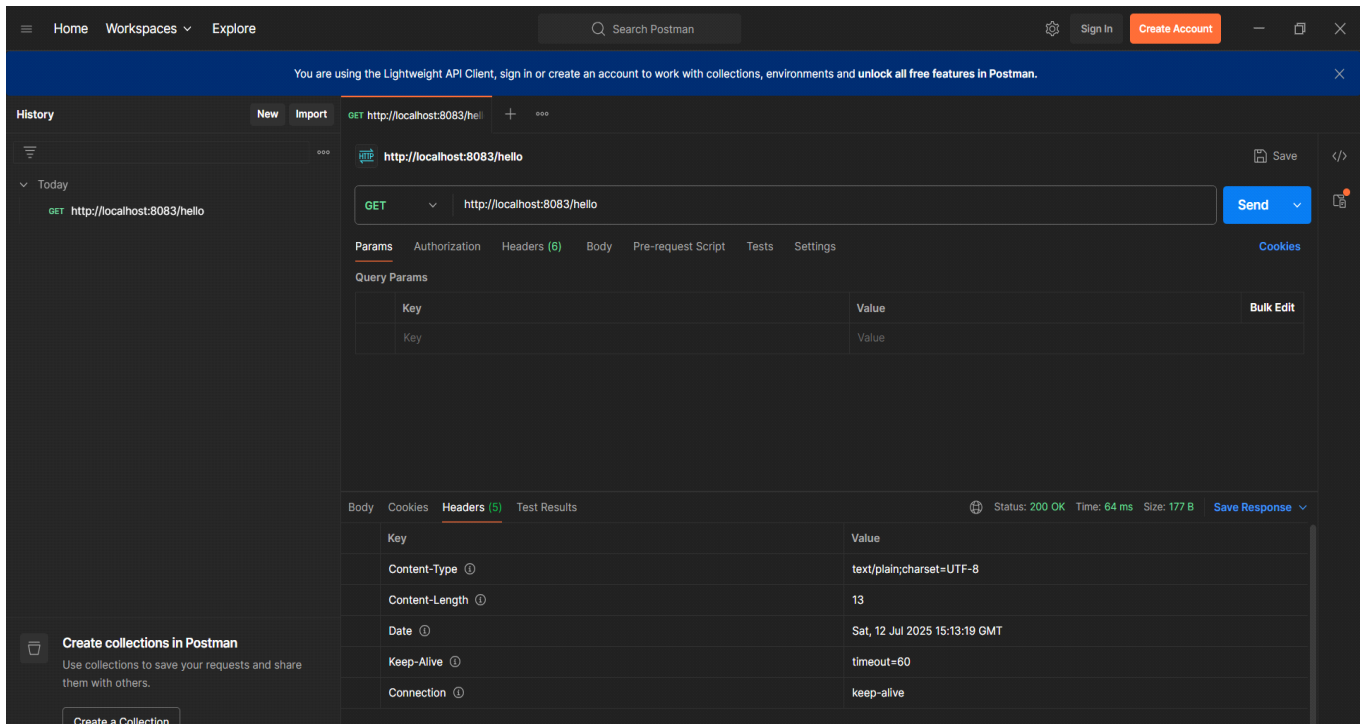
Browser :



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"
}
```

SME to explain the following aspects:

- What happens in the controller method?
- How the bean is converted into JSON response?
- 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

Country.java

```
package com.cognizant.spring_learn.model;

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

    public Country() {
    }

    public Country(String code, String name) {
        this.code = code;
        this.name = name;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

country.xml

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

    <bean id="country" class="com.cognizant.spring_learn.model.Country">
        <property name="code" value="IN" />
        <property name="name" value="India" />
    </bean>

</beans>
```

CountryController.java

```
package com.cognizant.spring_learn.Controller;

import com.cognizant.spring_learn.model.Country;
import org.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("country");
        LOGGER.info("END");
        return country;
    }
}

```

SpringLearnApplication.java

```

package com.cognizant.spring_learn;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

```

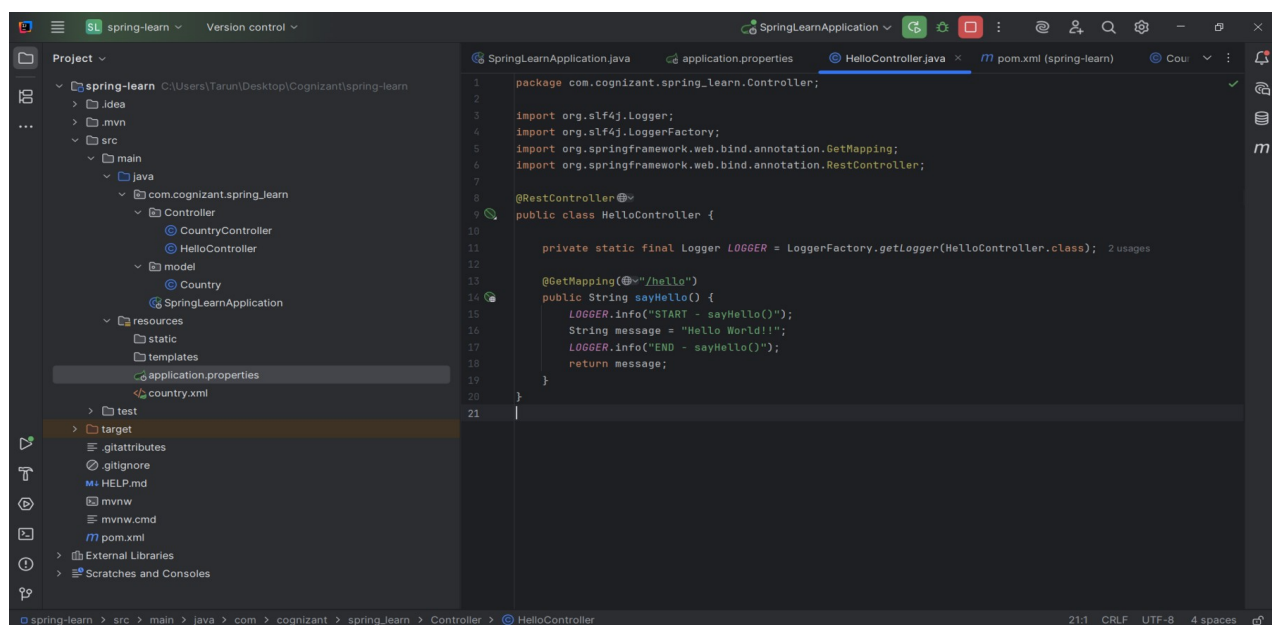
application.properties

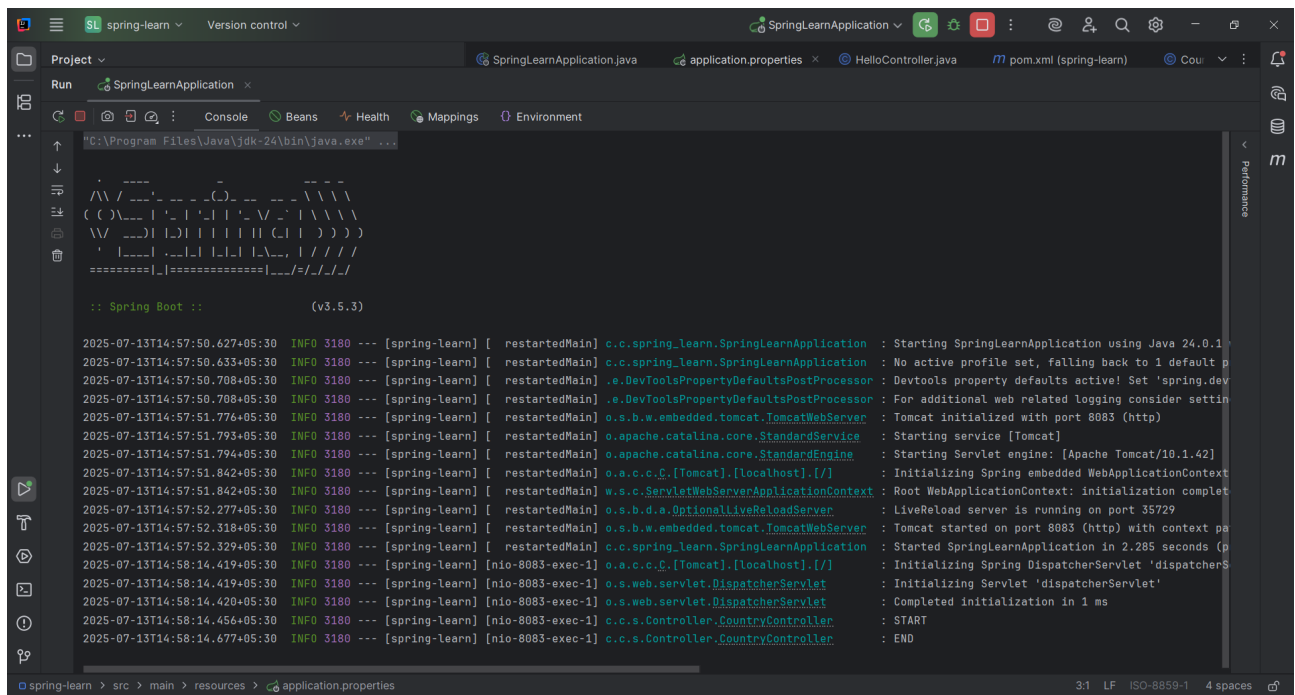
```

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

```

OUTPUT :





```

:: Spring Boot ::                (v3.5.3)

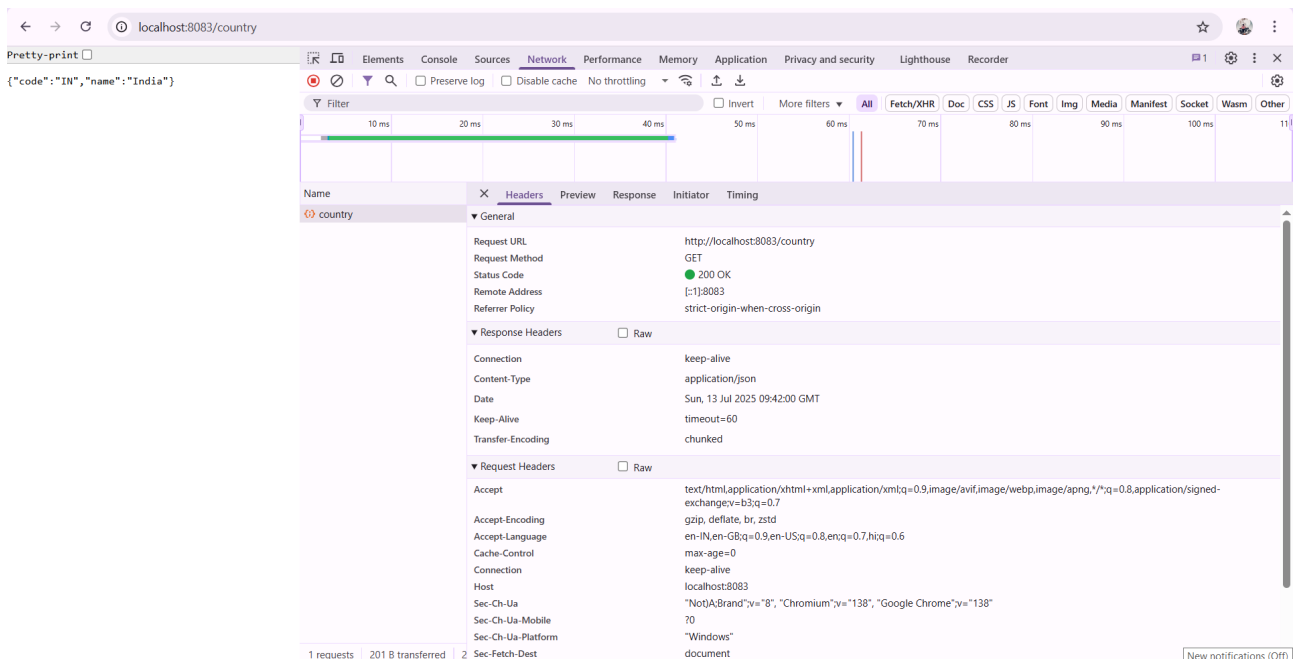
2025-07-13T14:57:50.627+05:30 INFO 3180 --- [spring-learn] [ restartedMain] c.c.spring_learn.SpringLearnApplication : Starting SpringLearnApplication using Java 24.0.1
2025-07-13T14:57:50.633+05:30 INFO 3180 --- [spring-learn] [ restartedMain] c.c.spring_learn.SpringLearnApplication : No active profile set, falling back to 1 default p
2025-07-13T14:57:50.708+05:30 INFO 3180 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.dev
2025-07-13T14:57:50.708+05:30 INFO 3180 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider settin
2025-07-13T14:57:51.776+05:30 INFO 3180 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8083 (http)
2025-07-13T14:57:51.793+05:30 INFO 3180 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-07-13T14:57:51.794+05:30 INFO 3180 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.42]
2025-07-13T14:57:51.842+05:30 INFO 3180 --- [spring-learn] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-07-13T14:57:51.842+05:30 INFO 3180 --- [spring-learn] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complet
2025-07-13T14:57:52.277+05:30 INFO 3180 --- [spring-learn] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2025-07-13T14:57:52.318+05:30 INFO 3180 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8083 (http) with context pa
2025-07-13T14:57:52.329+05:30 INFO 3180 --- [spring-learn] [ restartedMain] c.c.spring_learn.SpringLearnApplication : Started SpringLearnApplication in 2.285 seconds (p
2025-07-13T14:58:14.419+05:30 INFO 3180 --- [spring-learn] [nio-8083-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherS
2025-07-13T14:58:14.419+05:30 INFO 3180 --- [spring-learn] [nio-8083-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-07-13T14:58:14.420+05:30 INFO 3180 --- [spring-learn] [nio-8083-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2025-07-13T14:58:14.456+05:30 INFO 3180 --- [spring-learn] [nio-8083-exec-1] c.c.s.Controller.CountryController : START
2025-07-13T14:58:14.677+05:30 INFO 3180 --- [spring-learn] [nio-8083-exec-1] c.c.s.Controller.CountryController : END

```

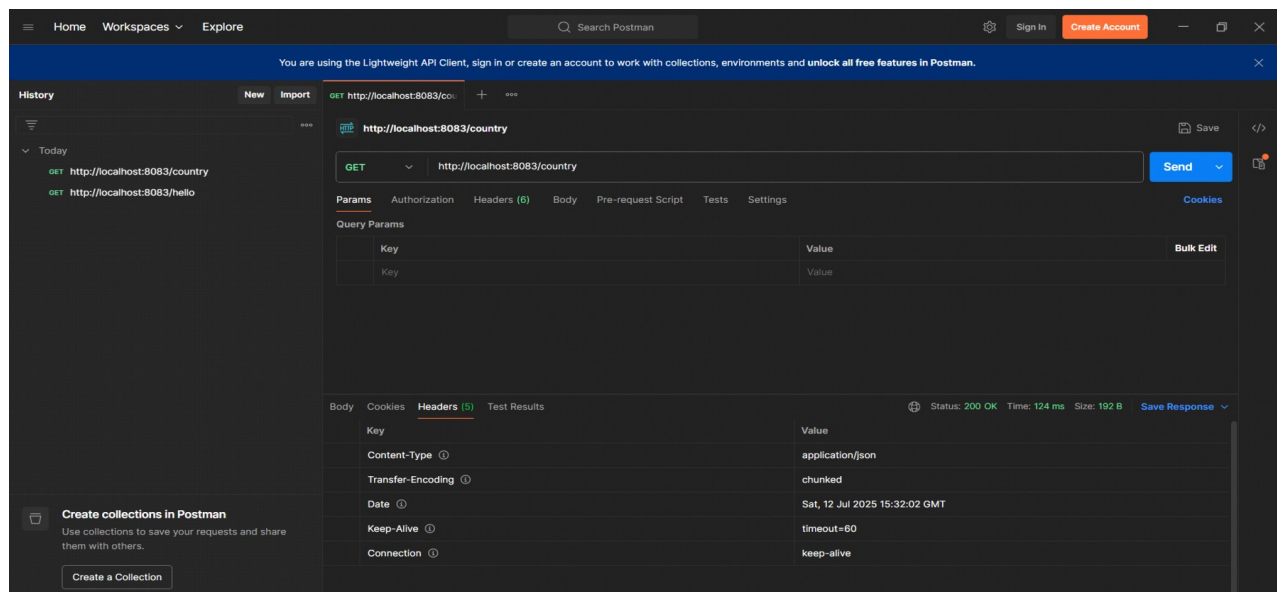
Browser:



- **What happens in the controller method?**
 - It loads ApplicationContext, fetches the country bean, and returns it.
- **How is the bean converted into JSON?**
 - Spring Boot uses Jackson internally to auto-convert Java objects to JSON when @RestController is used.
- **To check HTTP Headers:**
 - In browser: Open Dev Tools > Network tab > Click the /country request > Check “Headers” tab.



- In Postman: Click “Headers” tab after making the request.



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 Implementation: Invoke countryService.getCountry(code)

Service Method: com.cognizant.spring-learn.service.CountryService.getCountry(String code)

Service Method Implementation:

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

Sample Request: <http://localhost:8083/country/in>

Sample Response:

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

CODES :

Country .java

```
package com.cognizant.spring_learn.model;

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

    // Getters and Setters
    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code.toUpperCase(); // normalize
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

country.xml

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

    <bean id="us" class="com.cognizant.spring_learn.model.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>

```

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

        return countries.stream()
            .filter(country -> country.getCode().equalsIgnoreCase(code))
            .findFirst()
            .orElse(null); // You can throw exception here instead
    }
}

```

CountryController.java

```

package com.cognizant.spring_learn.controller;

import com.cognizant.spring_learn.model.Country;
import com.cognizant.spring_learn.service.CountryService;
import org.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()");
    Country country = countryService.getCountry(code);
    LOGGER.info("END getCountry()");
    return country;
}
```

application.properties

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

OUTPUT :

The screenshot displays an IDE interface with the following components:

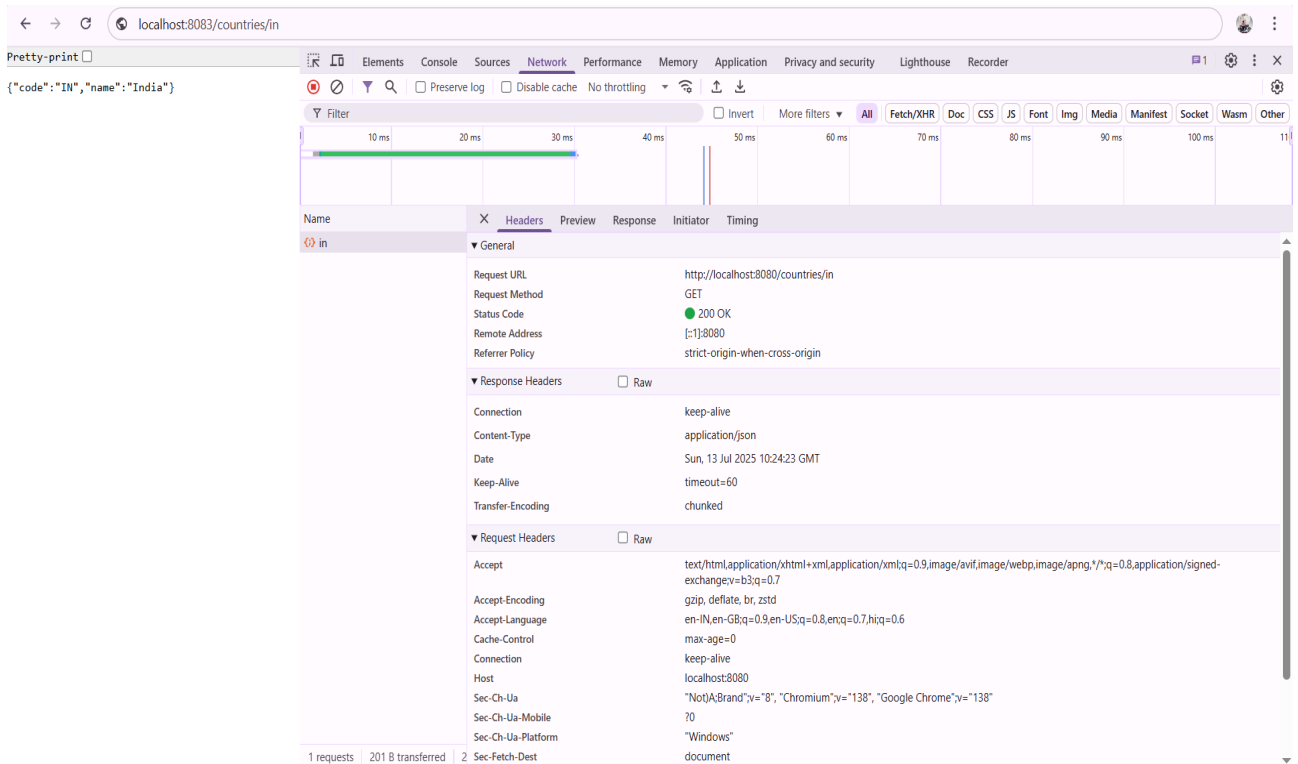
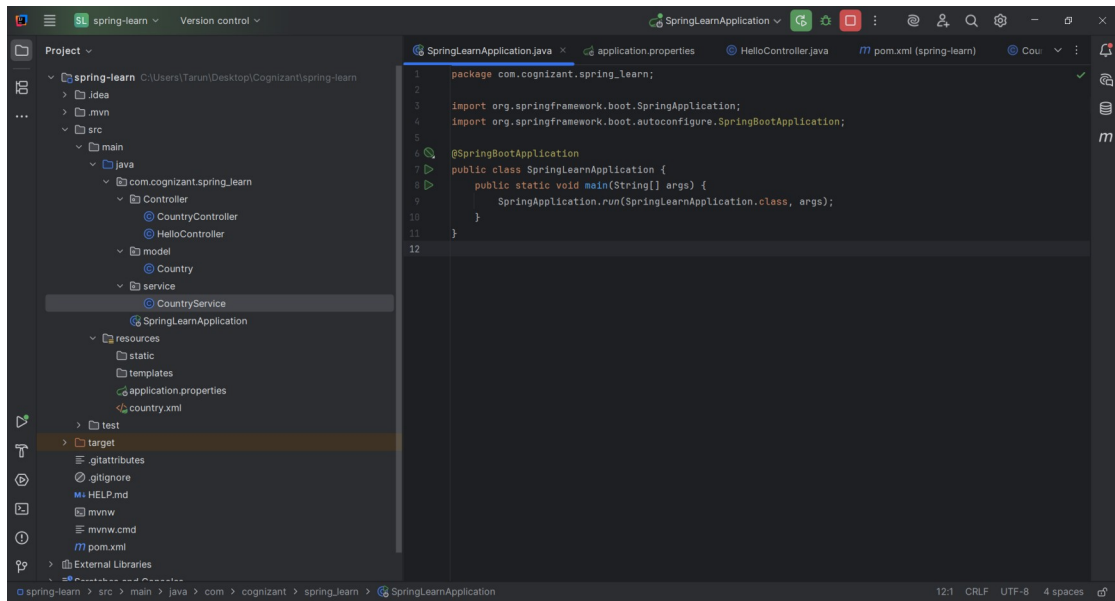
- Top Bar:** Includes icons for file operations, a search icon, and a window manager. The active window is titled "SpringLearnApplication.java".
- Project View:** Shows the project structure with "SpringLearnApplication.java" selected.
- Run Configuration:** The "Run" tab is active, showing the configuration for "SpringLearnApplication".
- Console:** Displays the output of the application. It starts with a ASCII art logo for "Spring Boot" (v3.5.3). The output shows the application starting successfully, using Java 24.0.1, and initializing the Spring framework and Tomcat web server. The application started in 2.314 seconds.
- Bottom Bar:** Shows the current file path: "spring-learn > src > main > java > com > cognizant > spring_learn > SpringLearnApplication". It also displays the current encoding as "UTF-8" and the line length as "121".

```

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  __/
      |_|_|_|

:: Spring Boot ::      (v3.5.3)

2025-07-13T15:36:21.904+05:30 INFO 4076 --- [spring-learn] [ restartedMain] c.c.spring_learn.SpringLearnApplication : Starting SpringLearnApplication using Java 24.0.1
2025-07-13T15:36:21.909+05:30 INFO 4076 --- [spring-learn] [ restartedMain] c.c.spring_learn.SpringLearnApplication : No active profile set, falling back to 1 default p
2025-07-13T15:36:21.984+05:30 INFO 4076 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.dev
2025-07-13T15:36:21.984+05:30 INFO 4076 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider settin
2025-07-13T15:36:23.048+05:30 INFO 4076 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-07-13T15:36:23.067+05:30 INFO 4076 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-07-13T15:36:23.068+05:30 INFO 4076 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.42]
2025-07-13T15:36:23.118+05:30 INFO 4076 --- [spring-learn] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-07-13T15:36:23.118+05:30 INFO 4076 --- [spring-learn] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complet
2025-07-13T15:36:23.576+05:30 INFO 4076 --- [spring-learn] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2025-07-13T15:36:23.616+05:30 INFO 4076 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context pa
2025-07-13T15:36:23.627+05:30 INFO 4076 --- [spring-learn] [ restartedMain] c.c.spring_learn.SpringLearnApplication : Started SpringLearnApplication in 2.314 seconds (p
```

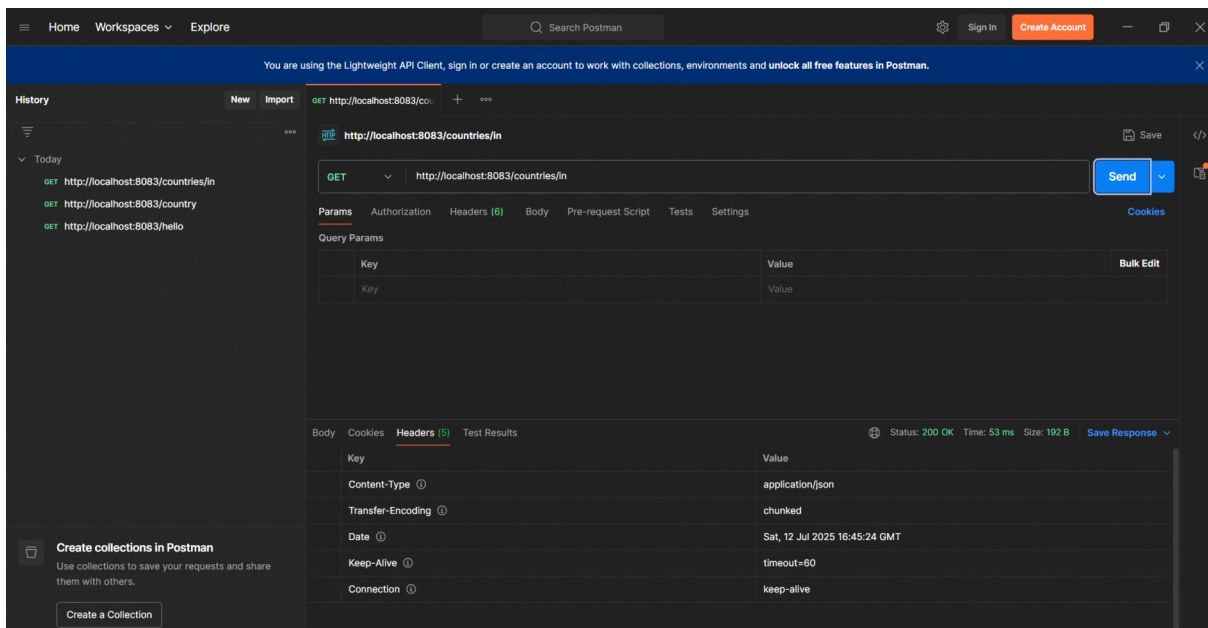


1 requests

201 B transferred

2 Sec-Fetch-Dest

document



JWT Hands on :

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.

CODES :

AuthenticationController.java

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

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

import java.util.HashMap;
import java.util.Map;

@RestController
public class AuthenticationController {

    @GetMapping("/authenticate")
    public Map<String, String> authenticate() {
        Map<String, String> response = new HashMap<>();
        response.put("token", "dummy.jwt.token");
        return response;
    }
}
```

SecurityConfig.java

```
package com.cognizant.spring_learn;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .httpBasic();
    }
}
```

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>
<description>Spring Learn REST Web Service Project</description>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.7.18</version>
  <relativePath/>
</parent>

<properties>
  <java.version>11</java.version>
</properties>

<dependencies>
  <!-- Web -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <!-- Security -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>

  <!-- JWT -->
  <dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
  </dependency>

  <!-- Devtools -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
  </dependency>

  <!-- Test -->
  <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>
```

OUTPUT:

```
* Host localhost:8090 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
* Trying [::1]:8090...
* Connected to localhost (::1) port 8090
* using HTTP/1.x
* Server auth using Basic with user 'user'
> GET /authenticate HTTP/1.1
> Host: localhost:8090
> Authorization: Basic dXNlcjpwd2Q=
> User-Agent: curl/8.13.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200
< Set-Cookie: JSESSIONID=DC387E4908B6864B501E4E51036131E0; Path=/; HttpOnly
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Pragma: no-cache
< Expires: 0
< X-Frame-Options: DENY
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Sat, 12 Jul 2025 17:46:07 GMT
<
★ {"token":"dummy-jwt-token"}* Connection #0 to host localhost left intact
```

localhost:8090/authenticate

dummy-jwt-token