**1.Spring-Rest-Handson**

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

**Objective**

The purpose of this hands-on task is to create and configure a basic Spring Boot web application using Maven. The task includes setting up the project using Spring Initializr, importing it into an IDE, configuring logging, building the project using Maven, and understanding the structure and configuration files used in the project.

**Step-by-Step Instructions**

**1. Generate Project Using Spring Initializr**

Visit <https://start.spring.io> and fill in the following fields:

* **Group**: com.cognizant
* **Artifact**: spring-learn
* **Name**: spring-learn
* **Description**: Spring Boot Hands-on Project
* **Packaging**: Jar
* **Java Version**: 17
* **Dependencies**:
  + Spring Web
  + Spring Boot DevTools

Click **Generate** to download the project as a ZIP file. Extract it to a suitable location (e.g., D:\spring-learn).

**2. Import the Project into IntelliJ**

* Open IntelliJ IDEA
* Select **File > New > Project from Existing Sources**
* Choose the extracted project folder
* Select **Maven** as the external model
* Click **Finish**

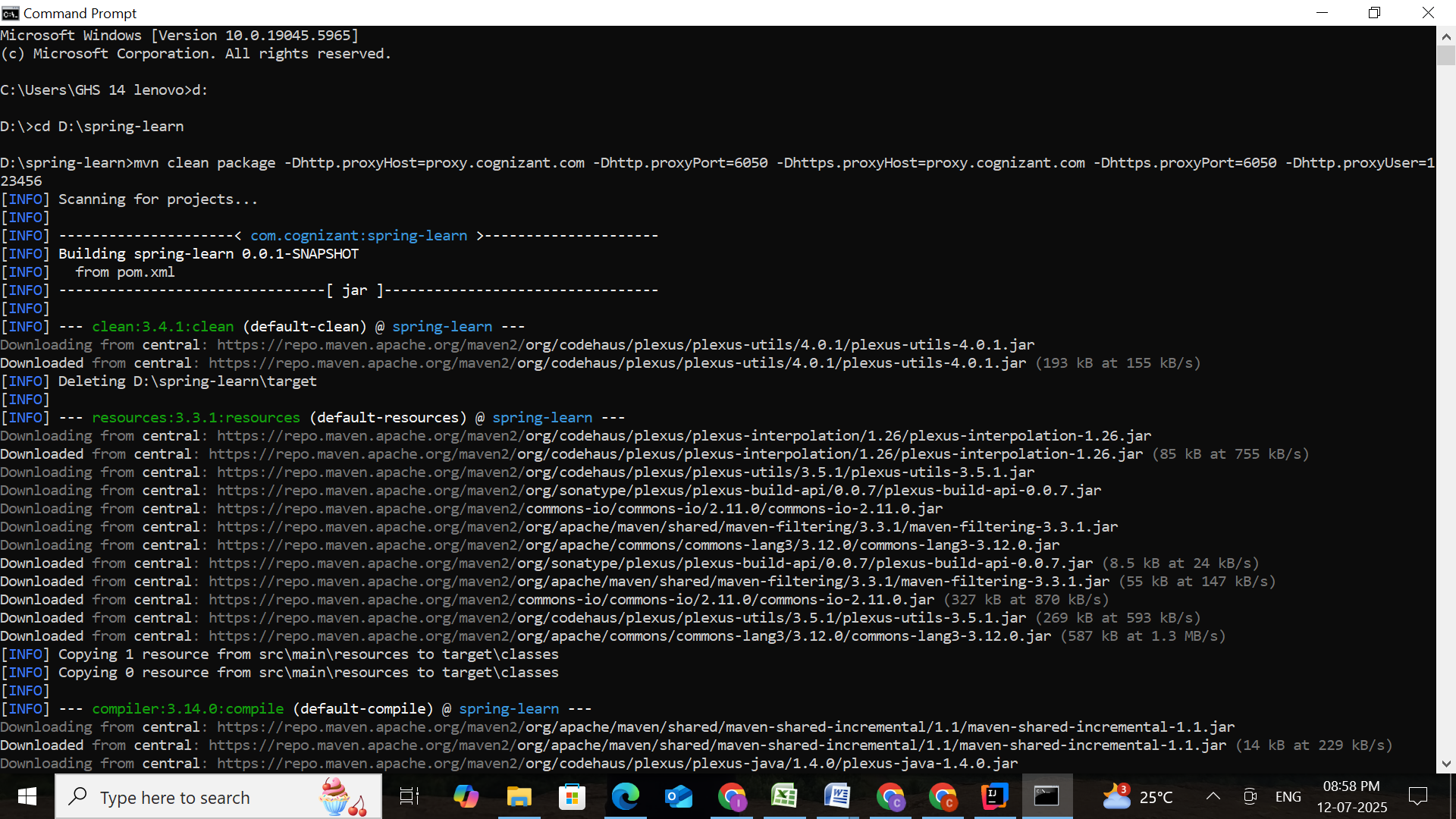
**3. Build the Project Using Maven**

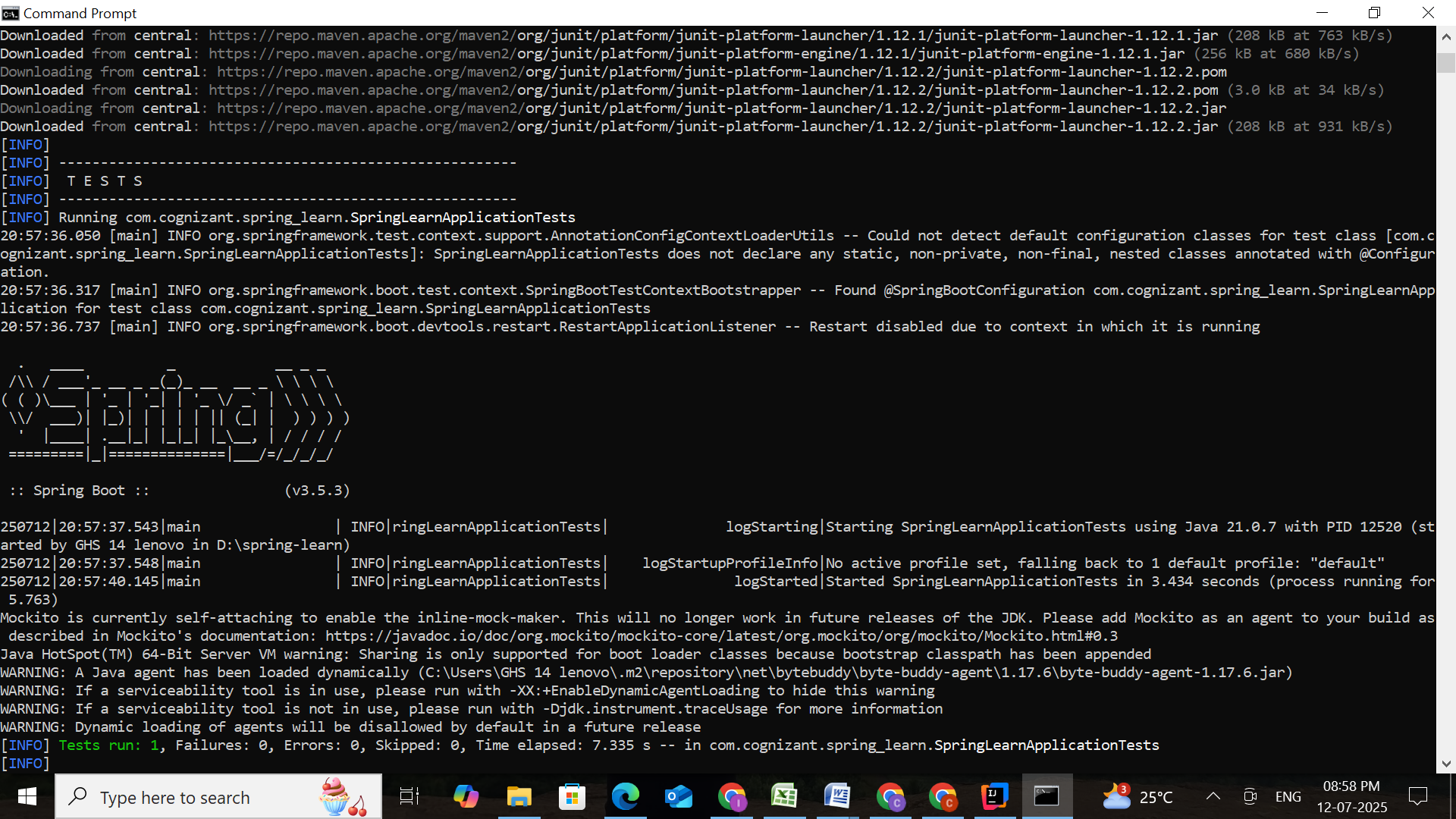
Run the following in Command Prompt :

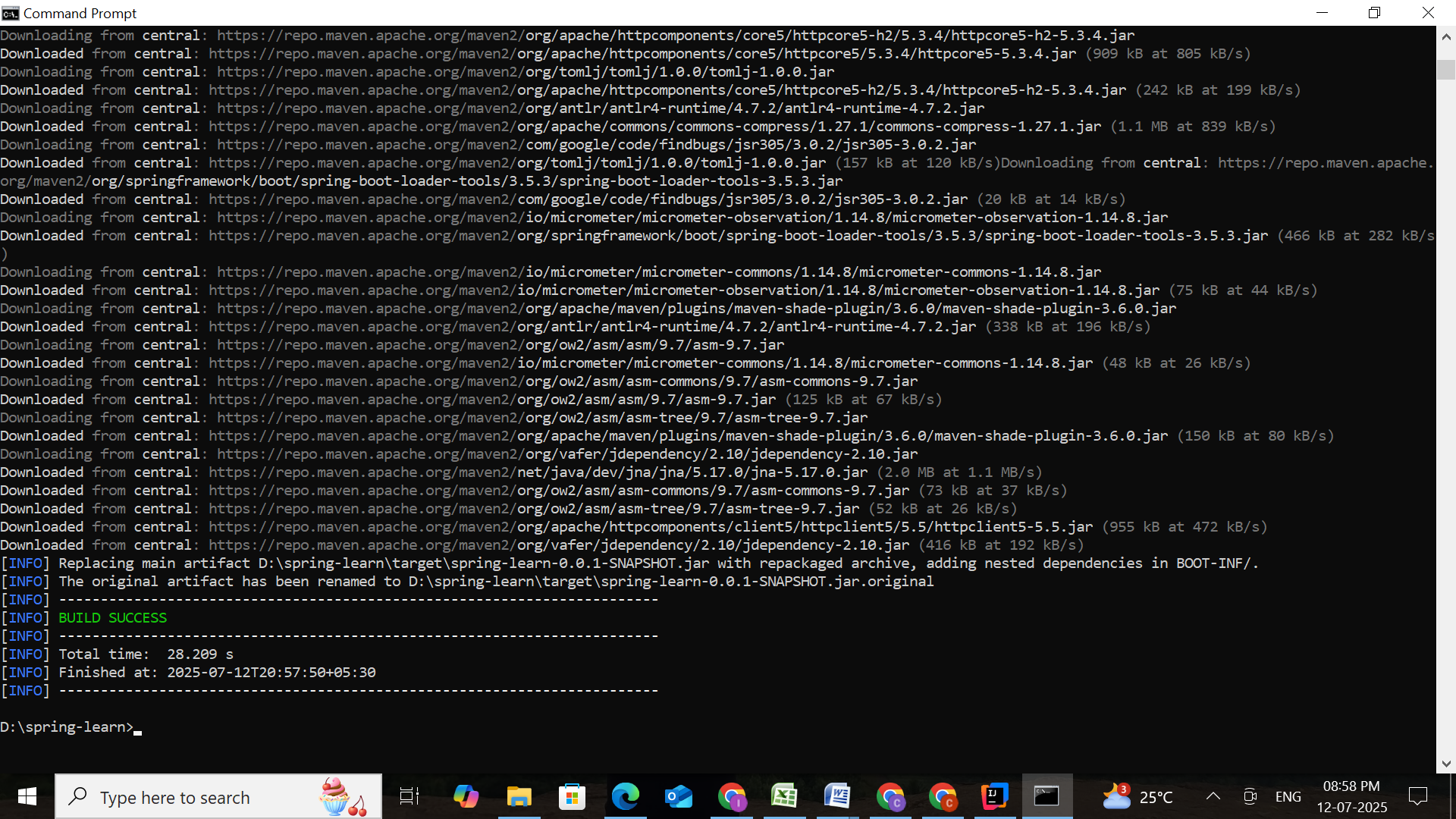
**mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456**

Once the build is complete, you should see:

**[INFO] BUILD SUCCESS**







**4. Add Logging Configuration**

Create a file named application.properties inside the src/main/resources folder. Add the following configuration to enable logging:

spring.application.name=spring-learn  
logging.level.org.springframework=info  
logging.level.com.cognizant.spring\_learn=debug  
logging.pattern.console=%d{yyMMdd}|%d{HH:mm:ss.SSS}|%-20.20thread|%5p|%-25.25logger{25}|%25M|%m%n

**5. Update the Main Class**

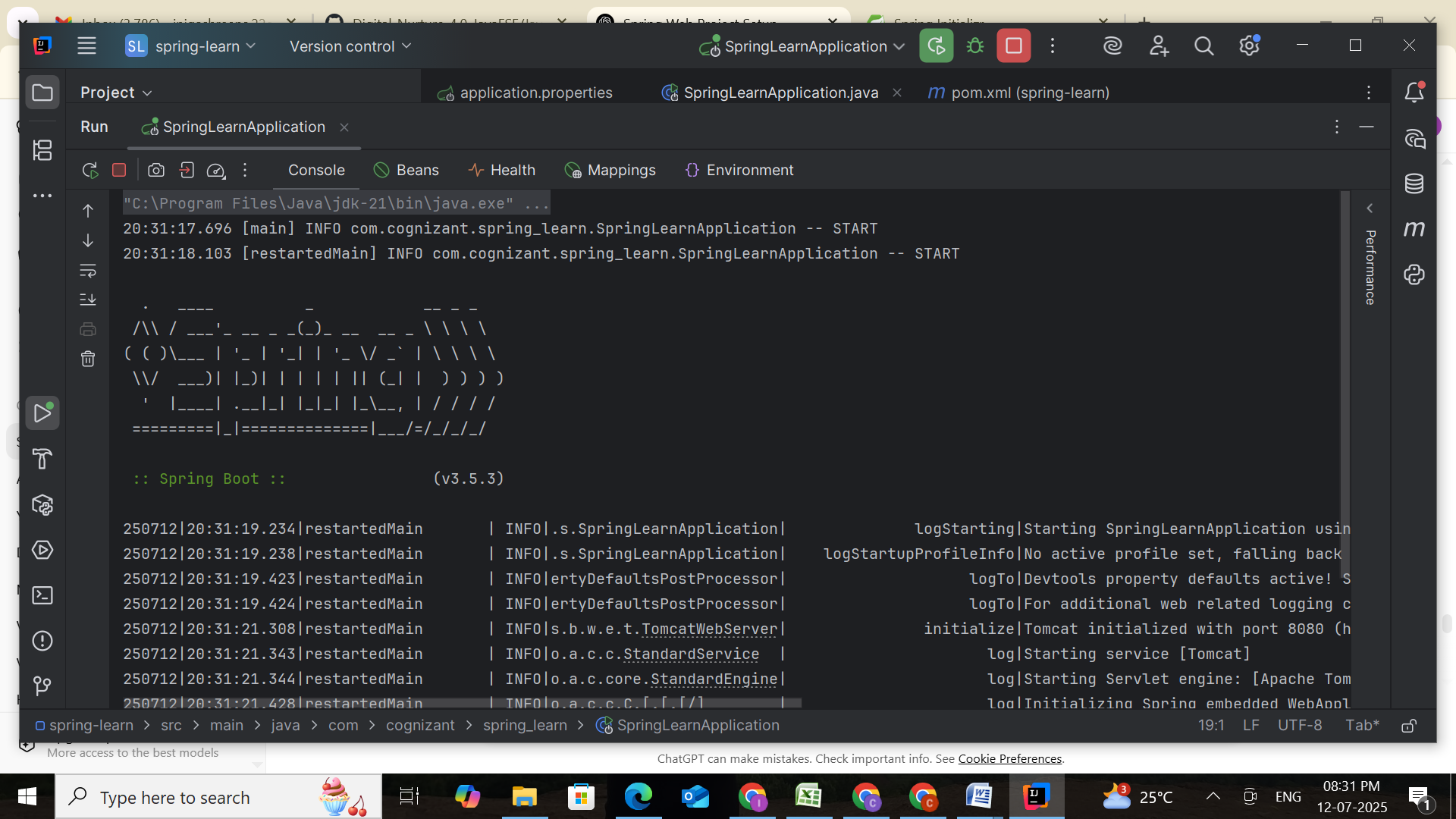
Open the file SpringLearnApplication.java inside src/main/java/com/cognizant/springlearn/. Update the file to include SLF4J logging.

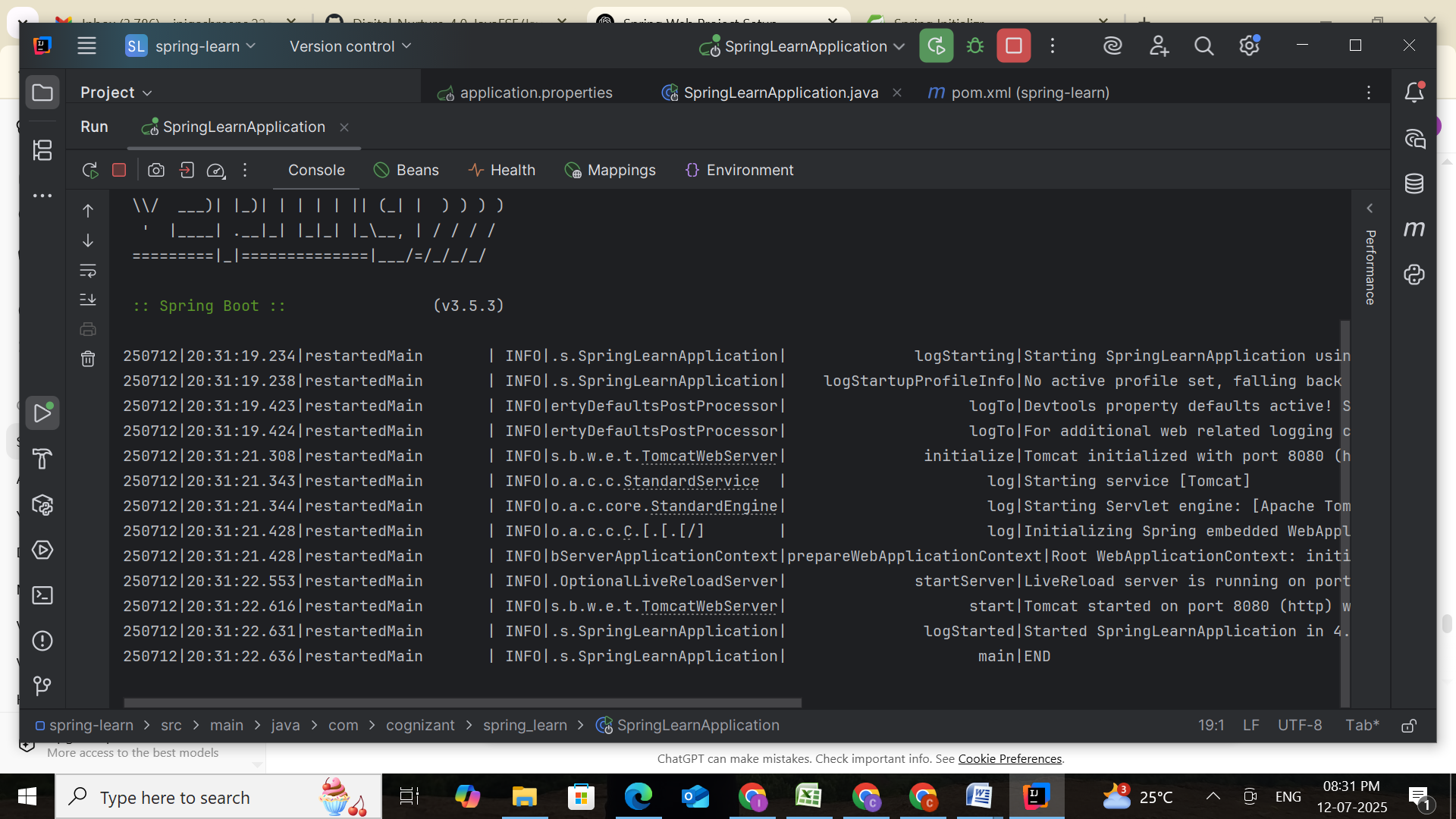
package com.cognizant.spring\_learn;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
@SpringBootApplication  
public class SpringLearnApplication {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(SpringLearnApplication.class);  
  
 public static void main(String[] args) {  
 *LOGGER*.info("START");  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 *displayCountry*();  
 *LOGGER*.info("END");  
 }

}

Right-click the class and run it as a Java Application.

**Output:**





**SME Walkthrough**

**Source Structure Explanation**

* **src/main/java**  
  Contains the Java source files, including the SpringLearnApplication.java main class.
* **src/main/resources**  
  Used to store configuration files such as application.properties.
* **src/test/java**  
  Contains the test files. This folder is created automatically by Spring Initializer.

**Main Class and Annotations**

* **Class**: SpringLearnApplication  
  Contains the main() method which is the entry point for the Spring Boot application.
* **@SpringBootApplication Annotation**  
  A convenience annotation that combines:
  + **@Configuration:** Declares the class as a configuration class.
  + **@EnableAutoConfiguration:** Enables Spring Boot’s auto-configuration feature.
  + **@ComponentScan:** Enables component scanning on the package.

**pom.xml Explanation**

The pom.xml file is the Maven configuration file that includes:

* **Parent Configuration**:

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

* **Dependencies**:

<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>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 </dependency>  
</dependencies>

* **Plugin Configuration**:

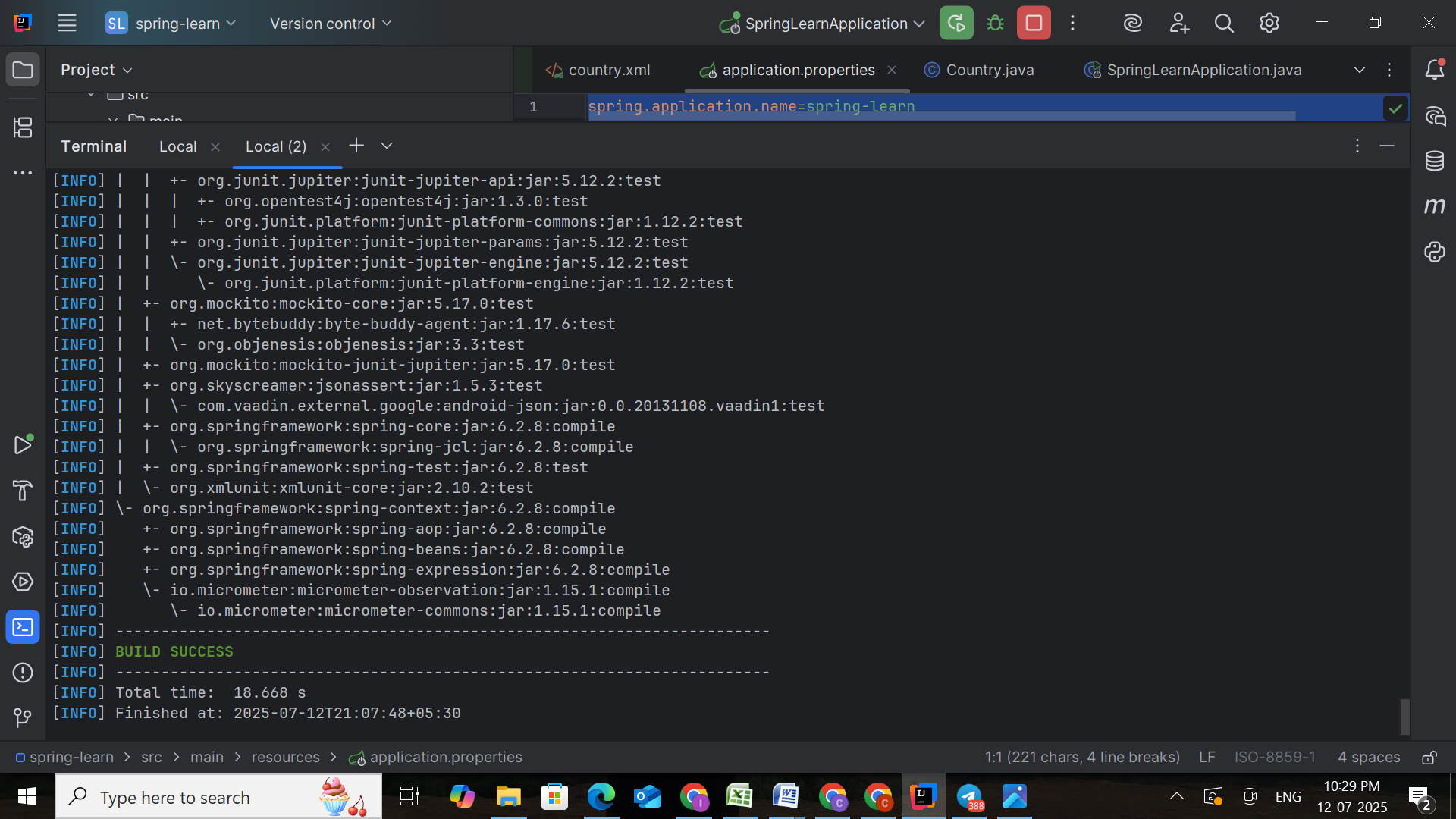
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

**Dependency Tree Verification**

Run the following command in terminal:

**mvn dependency:tree**

This command displays all dependencies and their transitive dependencies, helping identify what libraries are included in the build.



**Conclusion**

At the end of this hands-on, a fully functional Spring Boot web application is created using Maven. The project uses SLF4J for logging, follows proper Spring Boot conventions, and demonstrates a clean separation of source code and configuration files. This provides the foundational setup for upcoming tasks involving Spring XML configuration and bean injection.

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

**Objective**

An airline website is planning to support bookings from four countries. The homepage of the website will feature a dropdown to allow users to select a country. Each country must be represented using a two-letter ISO code and its full name. This information will be configured in an XML file using Spring Core.

**Country Configuration Data**

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

**Task Description**

Write a Spring Core program that reads one country bean from an XML configuration file and displays its details through a Java class. Logs should be printed for each constructor and method call to verify Spring's dependency injection process.

**Step-by-Step Implementation**

**Step 1: Create country.xml**

Location: **src/main/resources/country.xml**

Define a single country bean (e.g., India):

<?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>

**Step 2: Create the Country Class**

Location: **com.cognizant.spring\_learn.Country**

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 + '\'' +  
 '}';  
 }  
}

**Step 3: Update SpringLearnApplication.java**

Location: **com.cognizant.spring\_learn.SpringLearnApplication**

package com.cognizant.spring\_learn;  
  
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 main(String[] args) {  
 *LOGGER*.info("START");  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 *displayCountry*();  
 *LOGGER*.info("END");  
 }  
  
 public static void displayCountry() {  
 *LOGGER*.info("START");  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 Country country = context.getBean("country", Country.class);  
 *LOGGER*.debug("Country : {}", country.toString());  
 *LOGGER*.info("END");  
 }  
}

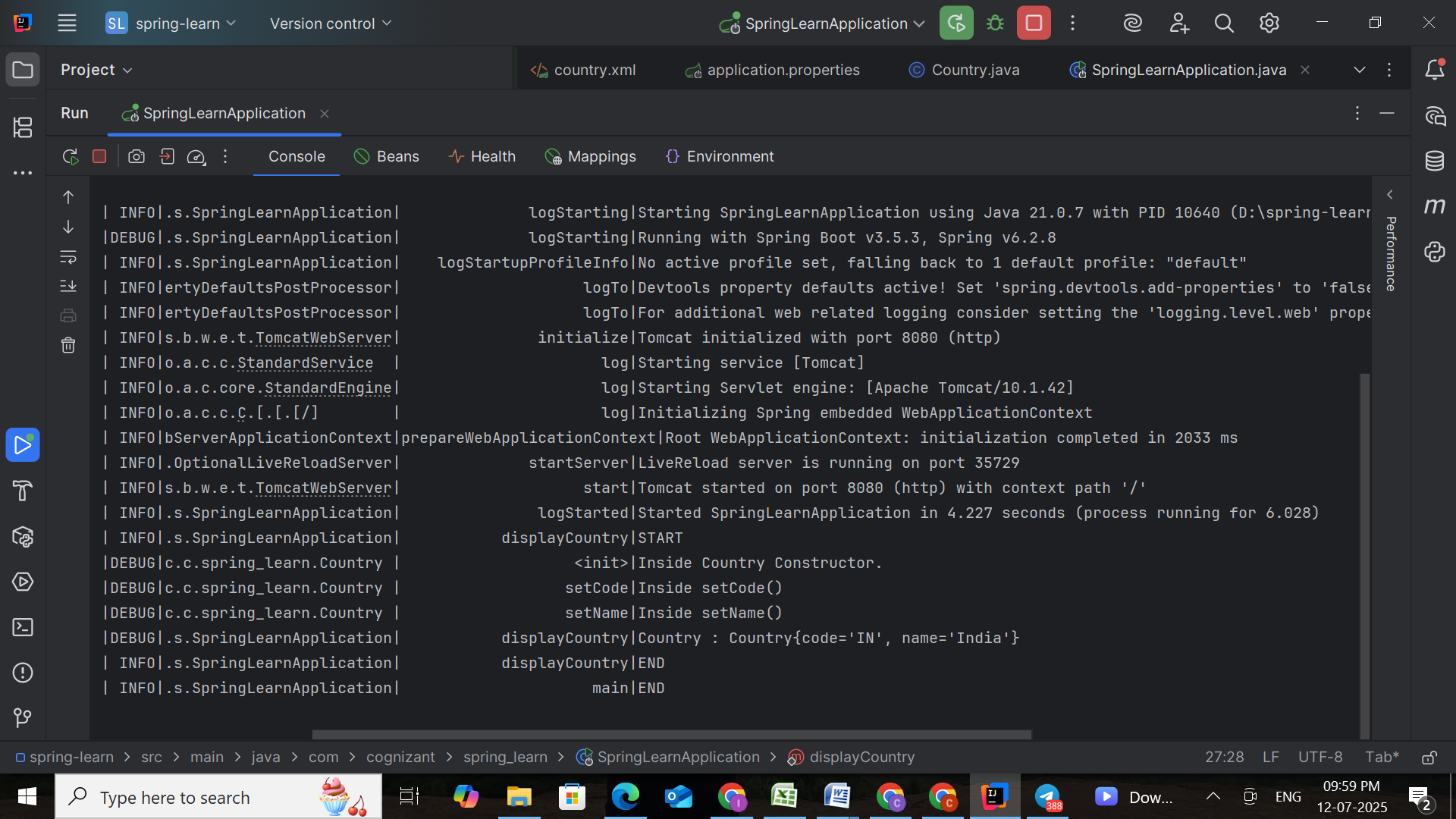
**Step 4: application.properties (for logging)**

Location: **src/main/resources/application.properties**

spring.application.name=spring-learn  
logging.level.org.springframework=info  
logging.level.com.cognizant.spring\_learn=debug  
logging.pattern.console=%d{yyMMdd}|%d{HH:mm:ss.SSS}|%-20.20thread|%5p|%-25.25logger{25}|%25M|%m%n

**Step 5: Output Validation**

When the application runs, the console should display logs like:



**SME Detailed Explanation**

**1. <bean>, id, class, <property>, name, value**

* <bean>: Declares an object to be managed by Spring.
* id: The reference name used to retrieve the bean.
* class: Full package-qualified name of the bean's Java class.
* <property>: Used to set properties using setter injection.
* name: The name of the property (e.g., code, name) — must match the setter method.
* value: The actual value to be injected into the property.

**2. ApplicationContext and ClassPathXmlApplicationContext**

* **ApplicationContext**: Spring's IoC container responsible for managing the lifecycle and configuration of beans.
* **ClassPathXmlApplicationContext:** Loads the Spring configuration from the class path (like src/main/resources/country.xml).

**3. What Happens When context.getBean() Is Invoked**

* Spring reads the bean definition from XML.
* Instantiates the bean using its default constructor.
* Injects the property values using the <property> tags and corresponding setter methods.
* Returns a fully constructed and configured bean object.