**Hands on 1**

**Create a Spring Web Project using Maven**

**SpringLearnApplication.java:**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

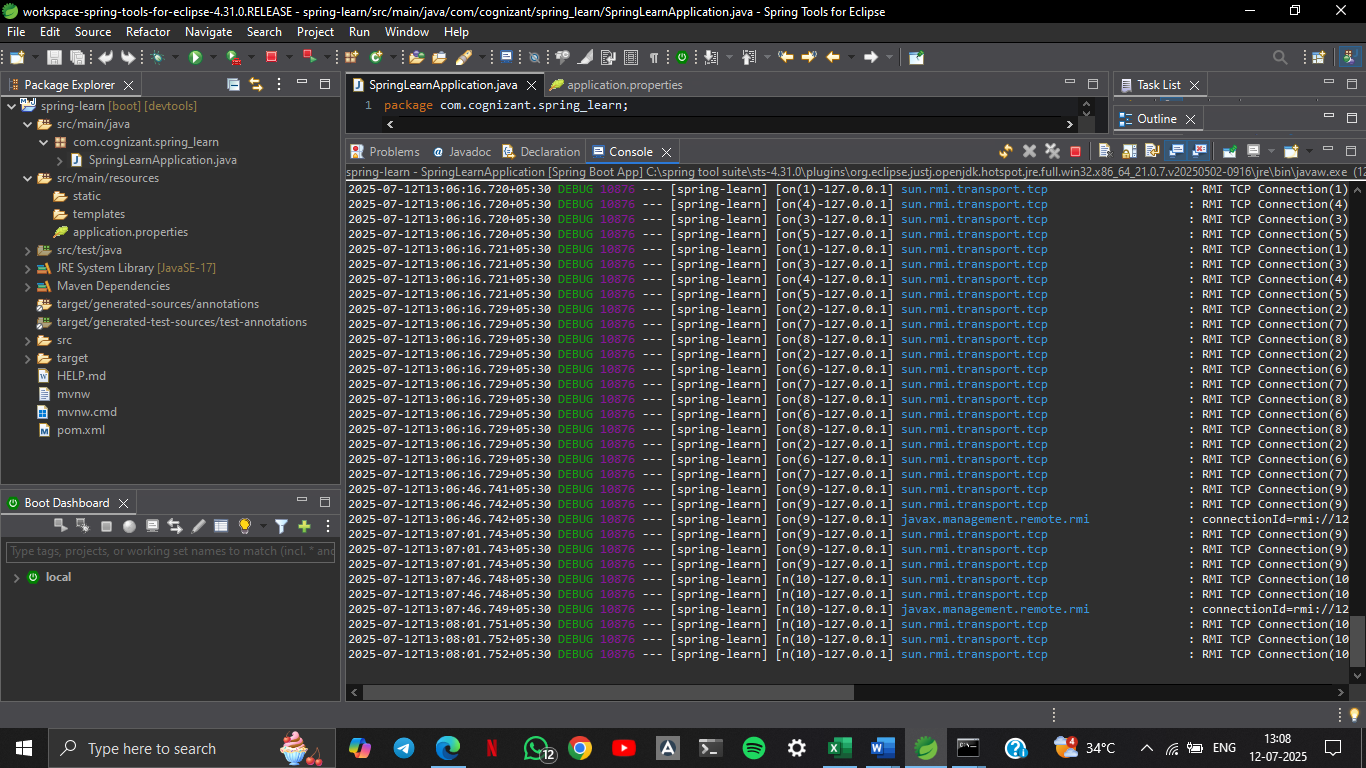
public static void main(String[] args) {

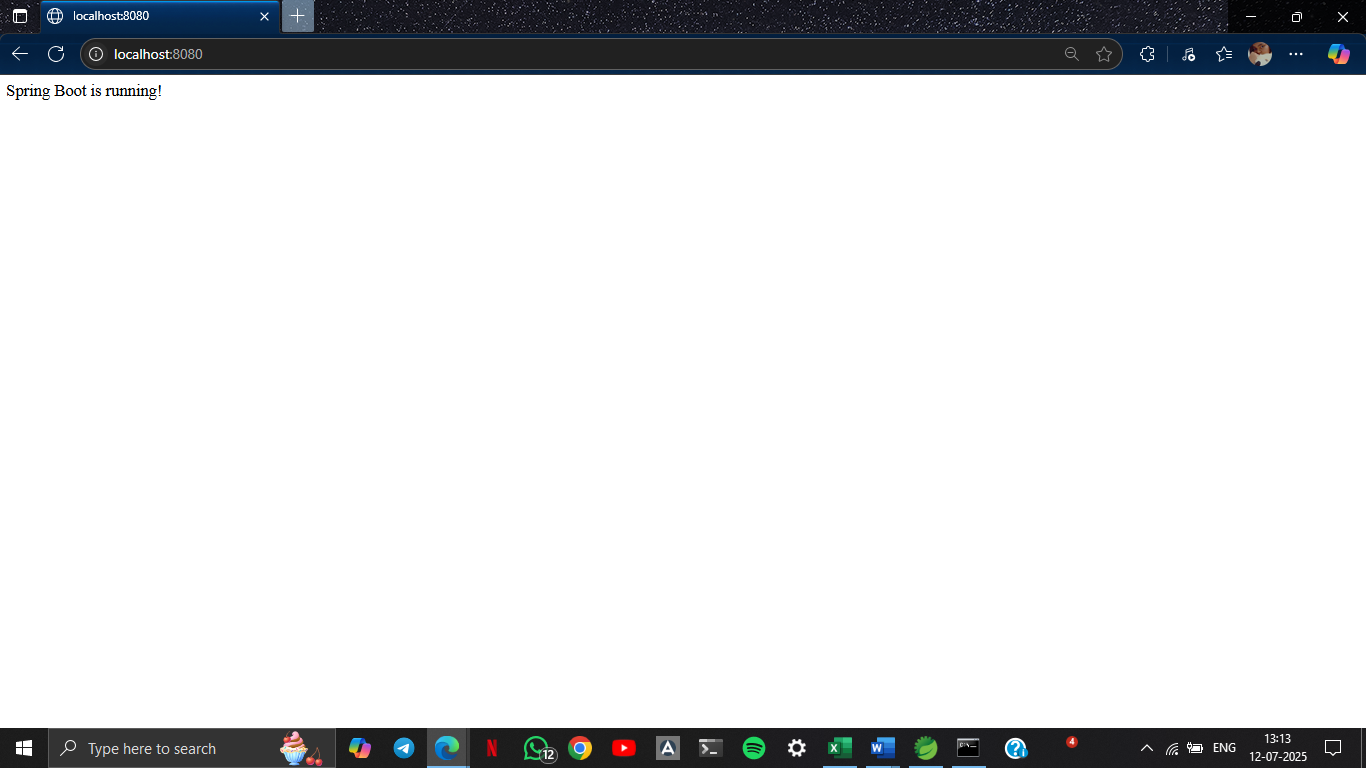
System.out.println("SpringLearnApplication is running...");

SpringApplication.run(SpringLearnApplication.class, args);

}

}

**OUTPUT:**



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

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

* Contains all the **main business logic**.
* SpringLearnApplication.java: The **main class** that launches the Spring Boot application.

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

* Stores all **non-code configuration files**:
* application.properties or application.yml: Used to configure port, database, logging, etc.
* country.xml: An **XML-based bean configuration** file (Spring Core style).

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

* Used to write **JUnit/TestNG-based test cases**.
* Spring Boot supports test configurations with:
* java
* CopyEdit
* @SpringBootTest
* Tests services, APIs, and repository layers.

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

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

* main() is the **entry point**.
* SpringApplication.run() bootstraps the entire application, scans for components, and starts the embedded **Tomcat** server by default.

1. **Purpose of @SpringBootApplication annotation**

It’s a **meta-annotation** that combines:

* @Configuration: Declares this as a Spring configuration class.
* @EnableAutoConfiguration: Automatically configures Spring beans based on classpath.
* @ComponentScan: Scans the package and sub-packages for beans (@Component, @Service, @RestController etc.)

1. **pom.xml**

Contains:

* Project metadata (groupId, artifactId, version)
* Dependencies (spring-boot-starter-web, spring-boot-devtools)
* Plugins (spring-boot-maven-plugin)
  1. Walkthrough all the configuration defined in XML file
  + <bean>: Declares a Spring Bean.
  + id: Unique name for accessing this bean.
  + class: Fully qualified class name.
  + <property>: Sets values into bean properties using setters.
  + <dependency>spring-boot-devtools</dependency> :Enables auto-restart, live reload, and better development experience
  + <parent>
  + <groupId>org.springframework.boot</groupId>
  + <artifactId>spring-boot-starter-parent</artifactId>
  + <version>3.5.3</version>
  + </parent>
  + Inherits default configurations from Spring Boot (like dependency versions, plugin config, etc.)
  + Makes version management easy — no need to declare versions for Spring dependencies manually.
  + <build>
  + <plugins>
  + <plugin>
  + <groupId>org.springframework.boot</groupId>
  + <artifactId>spring-boot-maven-plugin</artifactId>
  + </plugin>
  + </plugins>
  + </build>
  + Also packages your app as an executable .jar with embedded server.
  1. Open 'Dependency Hierarchy' and show the dependency tree.

