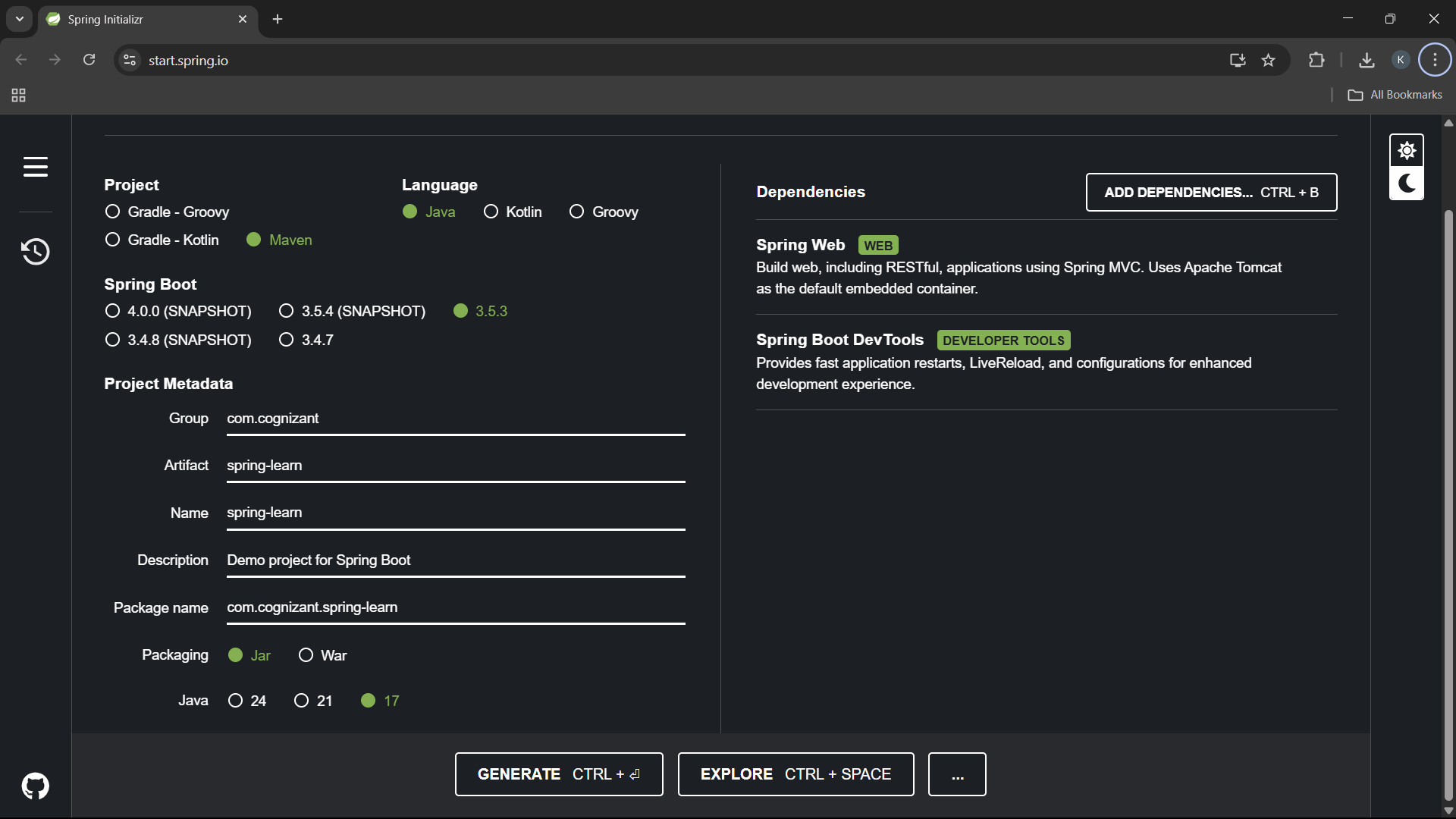
**Create a Spring Web Project using Maven**

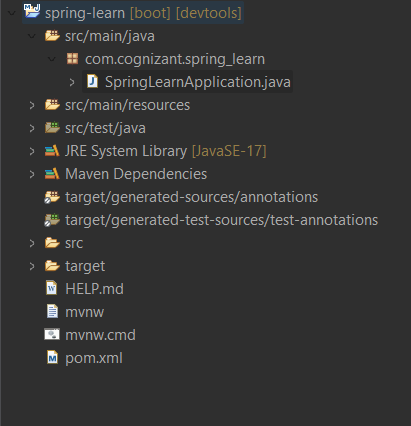
**Steps -**

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
5. Create and download the project as zip
6. Extract the zip in root folder to Eclipse Workspace
7. 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
8. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
9. Include logs to verify if main() method of SpringLearnApplication.
10. Run the SpringLearnApplication class.

**Spring Initializer –**



**Imported Project after extraction –**

****

**MainApplication –**

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);

System.out.println("SpringLearnApplication started...");

}

}

**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>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-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>

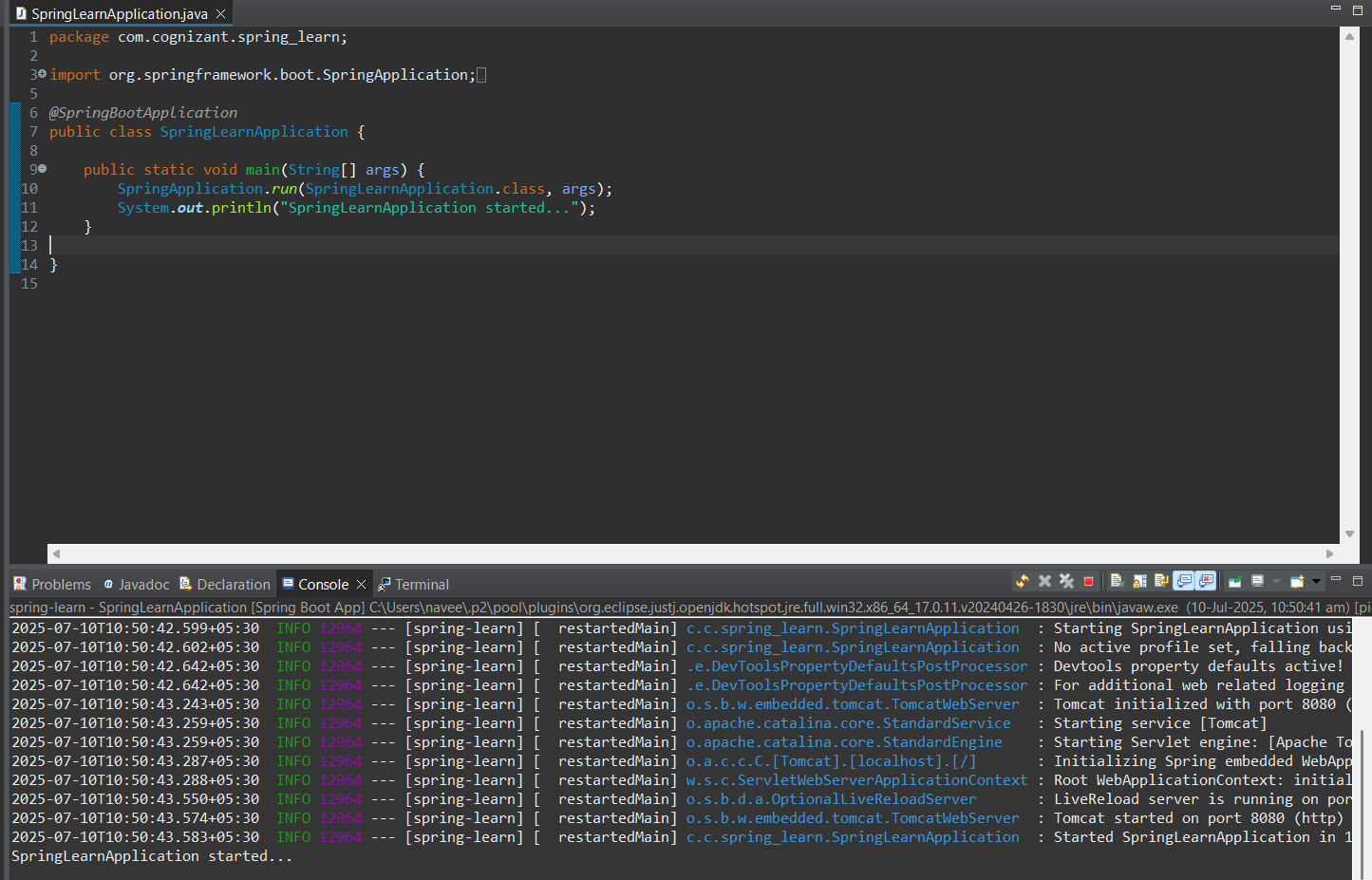
</plugin>

</plugins>

</build>

</project>

**Output :**

****

**SME –**

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

**Answer** - This directory contains all the **main Java source files** of the application. This is the core folder where we write the business logic, controllers, services, and other application classes.

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

**Answer -** This is where we keep all configuration and static resource files. These resources are automatically picked up by Spring Boot at runtime. Ex. Application.properties.

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

**Answer -**  Contains unit and integration test cases written using JUnit or Spring Boot Test. It mirrors the structure of src/main/java so each class can have a corresponding test class. Example: CountryTest.java would test the Country.java class.

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

**Answer -** This class contains the main() method which is the entry point of the application.

**Code –**

@SpringBootApplication

public class SpringlearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringlearnApplication.class, args);

}

}

**Inside main():**

* SpringApplication.run(...) bootstraps the Spring Boot application.
* It automatically scans all classes in the package and subpackages to register beans and components.

1. **Purpose of @SpringBootApplication annotation**

**Answer -** This is a composite annotation that includes:

* @Configuration: Allows the class to define beans using @Bean methods.
* @EnableAutoConfiguration: Automatically configures Spring components based on dependencies on the classpath.
* @ComponentScan: Scans for components like @RestController, @Service, @Component in the current package and subpackages.

This annotation is essential to enable Spring Boot’s auto-configuration and component scanning features.

1. **pom.xml**
   1. **Walkthrough all the configuration defined in XML file**

**Answer –**

*Parent Tag –*

<parent>

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

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

<version>3.5.3</version>

</parent>

* Inherits Spring Boot’s default dependency versions, plugin configuration, and dependency management.
* Ensures compatibility with Spring Boot 3.5.3 and simplifies version control for dependencies.

*Project Coordinates –*

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

These values uniquely identify your project in a Maven repository.

* groupId: Organization or company name
* artifactId: Project or module name
* version: Current version of the project (SNAPSHOT indicates development version)

*Java Version –*

<properties>

<java.version>17</java.version>

</properties>

* Specifies that the project should be compiled with **Java 17**.
* Used by Spring Boot to generate correct bytecode and avoid compatibility issues.

*Dependencies Section –*

This section tells Maven what libraries your project depends on.

1. Spring Boot Web Starter

<dependency>

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

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

</dependency>

Brings in everything required for building a RESTful web application:

* Spring MVC, Tomcat (embedded), Jackson (for JSON), logging, etc.

1. Spring Boot DevTools

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

* Adds auto-restart and hot-reloading support.
* Makes development faster by automatically restarting the app when files are changed.

1. Spring Boot Starter Test

<dependency>

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

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

<scope>test</scope>

</dependency>

Provides libraries like:

* **JUnit 5** for writing unit tests
* **Mockito** for mocking
* **Spring Test** for integration testing

*Build Section – Plugin –*

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

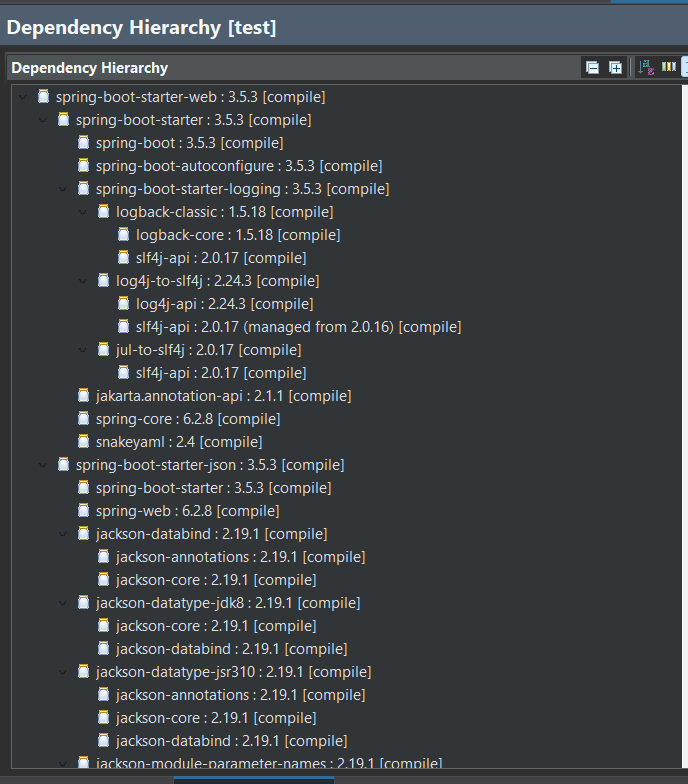
</build>

Allows you to **build and run** the application using:

* mvn spring-boot:run
* mvn package (generates an executable .jar)

Embeds the application server (Tomcat) into the JAR, so no external server is needed.

**Open 'Dependency Hierarchy' and show the dependency tree.**

****