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

# Objective:

To create a Spring Boot web project using Maven, import it into Eclipse, understand the project structure, and demonstrate basic execution with logging.

# Steps Followed

## Step 1: Project Creation using Spring Initializr

• Visited: https://start.spring.io

• Group: com.cognizant

• Artifact: spring-learn

• Selected Dependencies: Spring Web, Spring Boot DevTools

• Downloaded and extracted the project ZIP.

## Step 2: Import Project in Eclipse

• Extracted ZIP in Eclipse Workspace.

• Imported using: File > Import > Maven > Existing Maven Projects > Selected extracted folder > Finish

## Step 3: Build Project via Command Line

Used the following Maven build command:

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

(Note: Replace 123456 with actual proxy user ID)

## Step 4: Add Logging to Main Class

Edited the main class SpringLearnApplication.java as below:

package com.cognizant.springlearn;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
@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);  
 LOGGER.info("END");  
 }  
}

## Step 5: Run the Application

• Ran the application using: Right-click > Run As > Java Application

• Console Output:

INFO com.cognizant.springlearn.SpringLearnApplication - START  
...Spring Boot startup logs...  
INFO com.cognizant.springlearn.SpringLearnApplication - END

# Project Structure Walkthrough

• src/main/java – Contains Java source code for application logic.

• src/main/resources – Contains configuration files like application.properties.

• src/test/java – Contains test classes.

# SpringLearnApplication.java Overview

• Entry point of the application. Uses SpringApplication.run() to start embedded Tomcat.

• Used SLF4J Logger to print "START" and "END" messages.

# Understanding @SpringBootApplication Annotation

This annotation includes:

1. @Configuration – Declares this class as configuration.

2. @EnableAutoConfiguration – Enables Spring Boot auto config.

3. @ComponentScan – Scans packages for Spring components.

# pom.xml Overview

Basic Project Info:

<groupId>com.cognizant</groupId>  
<artifactId>spring-learn</artifactId>  
<version>0.0.1-SNAPSHOT</version>

Key Dependencies:

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

# Dependency Hierarchy

• Viewed using: Right-click project > Maven > Show Dependency Hierarchy

• spring-boot-starter-web includes Spring MVC, Embedded Tomcat, Jackson, Validation, etc.

• spring-boot-devtools enables live reload.

• spring-boot-starter-test adds testing tools.

# Conclusion

Successfully created a Spring Boot web project using Maven, imported to Eclipse, verified logging functionality, and understood project structure and configurations.