

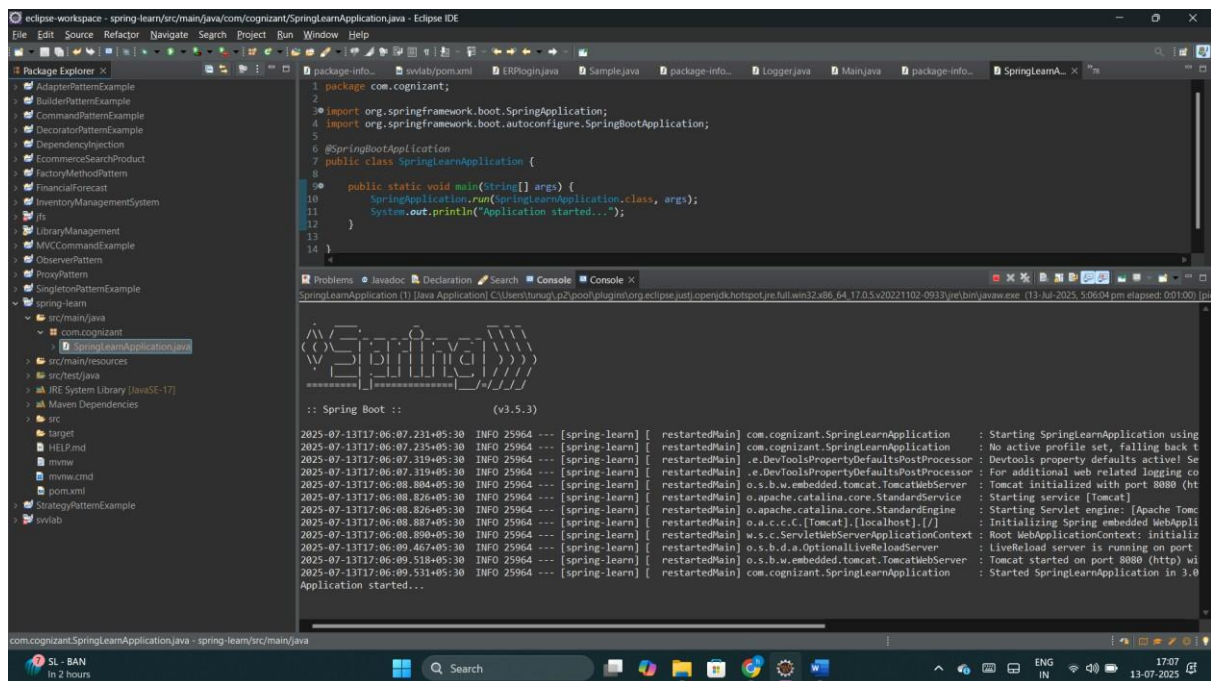
Name: Ilam Venkata Satwik  
Superset Id: 4992334  
Email : [2200032398cseh@gmail.com](mailto:2200032398cseh@gmail.com)

## Week 4 – 1. Spring-Rest-Handson:

1. Create spring web project using maven

Output:

Src/main/java and output



The screenshot shows the Eclipse IDE with the SpringLearnApplication.java file open. The code is as follows:

```
1 package com.cognizant;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class SpringLearnApplication {  
8  
9     public static void main(String[] args) {  
10         SpringApplication.run(SpringLearnApplication.class, args);  
11         System.out.println("Application started...");  
12     }  
13 }  
14 }
```

The console output shows the application starting successfully:

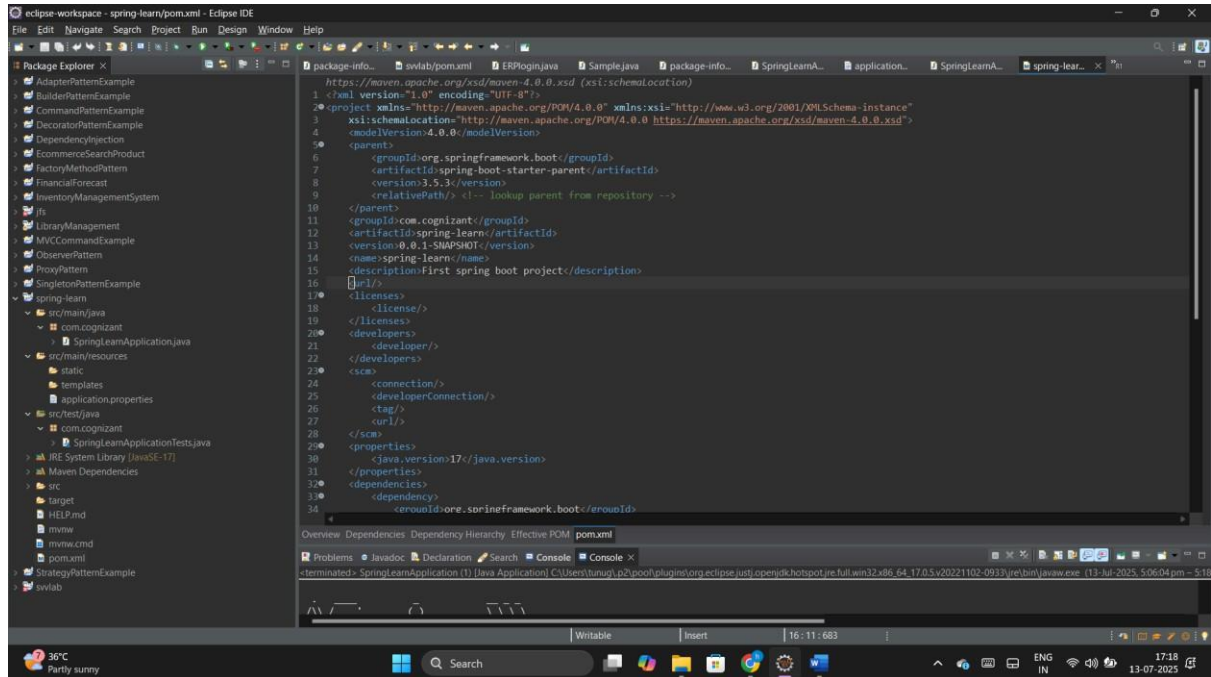
```
2025-07-13T17:06:07.221+05:30 INFO 25964 --- [spring-learn] [ restartedMain] com.cognizant.SpringLearnApplication : Starting SpringLearnApplication using  
2025-07-13T17:06:07.235+05:30 INFO 25964 --- [spring-learn] [ restartedMain] com.cognizant.SpringLearnApplication : No active profile set, falling back to default  
2025-07-13T17:06:07.319+05:30 INFO 25964 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : DevTools property defaults active. See  
2025-07-13T17:06:07.319+05:30 INFO 25964 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging co  
2025-07-13T17:06:08.894+05:30 INFO 25964 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (ht  
2025-07-13T17:06:08.826+05:30 INFO 25964 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]  
2025-07-13T17:06:08.826+05:30 INFO 25964 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomc  
2025-07-13T17:06:08.887+05:30 INFO 25964 --- [spring-learn] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebAppli  
2025-07-13T17:06:08.890+05:30 INFO 25964 --- [spring-learn] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializ  
2025-07-13T17:06:09.467+05:30 INFO 25964 --- [spring-learn] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port  
2025-07-13T17:06:09.518+05:30 INFO 25964 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http wi  
2025-07-13T17:06:09.531+05:30 INFO 25964 --- [spring-learn] [ restartedMain] com.cognizant.SpringLearnApplication : Started SpringLearnApplication in 3.0  
Application started...
```

Walk through of the code:

@SpringBootApplication is a shortcut annotation for:

1. @Configuration – configuration class
2. @EnableAutoConfiguration – enables Spring Boot auto-configuration
3. @ComponentScan – scans com.cognizant package and sub-packages for Spring components.

Pom.xml:



Header and Meta data:

Defines this file as a Maven POM (Project Object Model)

XML namespaces used to validate and understand Maven structure.

Parent configuration:

Inherits settings from Spring Boot's parent POM

Provides default plugin configurations and dependency versions Ensures consistency across Spring Boot projects

Project identity:

groupId: your company or domain name

artifactId: project name

version: version of the project (you can later change to 1.0.0 etc.)

Project description :

Used for project documentation or publishing to repositories

Java version –

Tells Spring Boot and Maven to compile using Java 17

Make sure your system has JDK 17 installed

Dependencies – webstarter , dev tools, unit testing : Adds

Spring MVC, embedded Tomcat, JSON support

Required for web/REST APIs,

Enables auto-restart and live reload during development Only runs

in dev mode (not in production builds),

Adds JUnit, Mockito, and other test libraries.

Dependency Hierarchy:

spring-boot-starter-web

└─ spring-boot-starter

└─ spring-boot

└─ spring-core

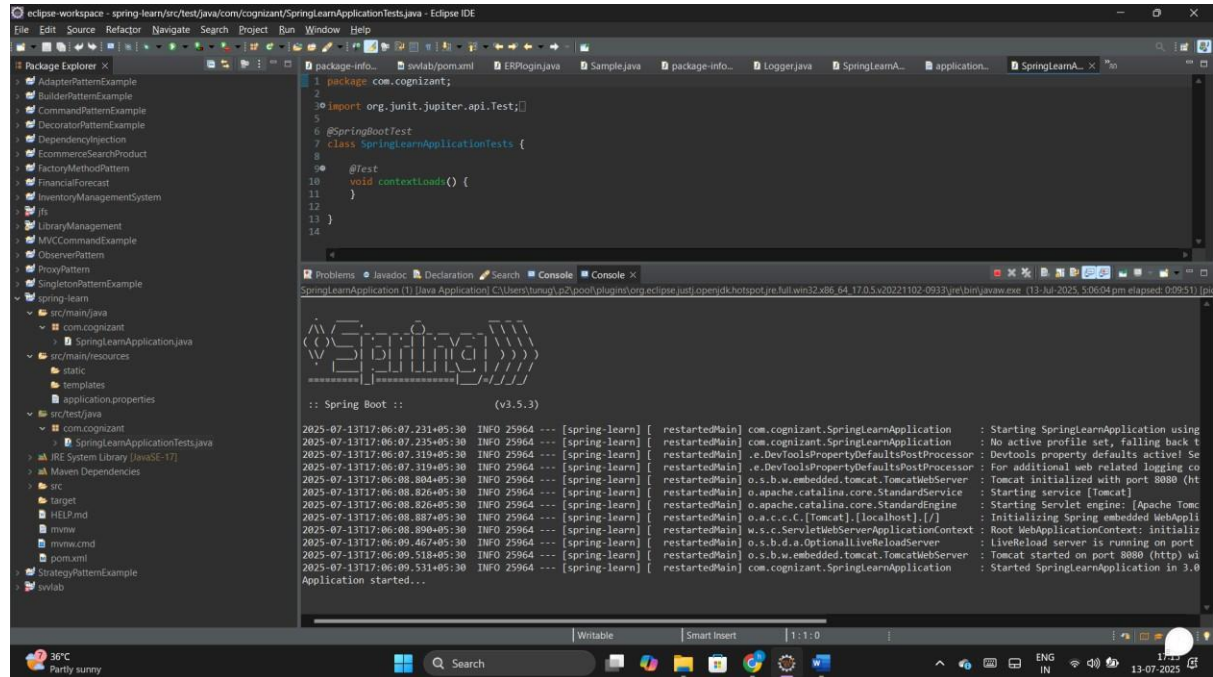
└─ ...

└─ spring-web

└─ spring-webmvc

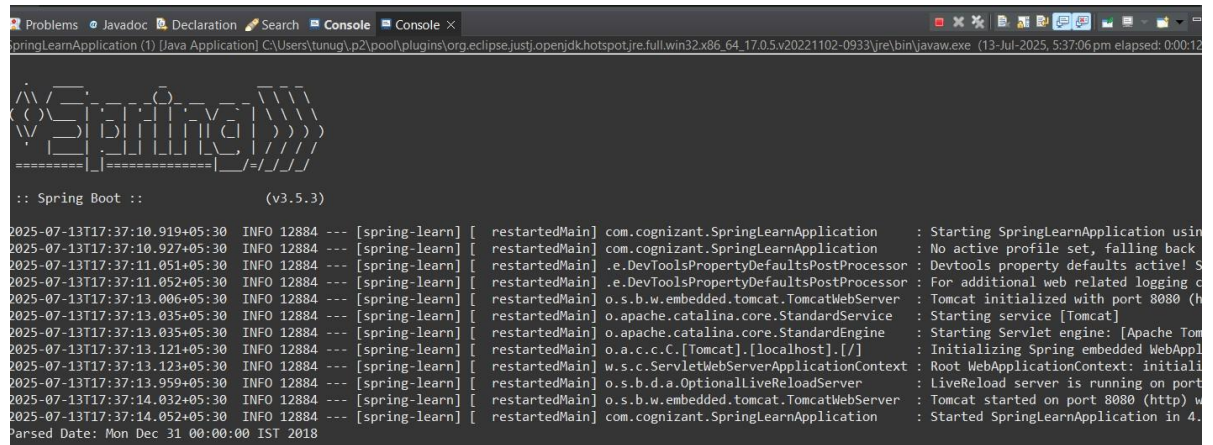
└─ tomcat-embed-core

## Sec/test/java:



## 2. Load SimpleDateFormat from Spring Configuration XML

Output:



## 2.SPRING-REST-HANDSON

### 1. Hello World RESTful Web Service :

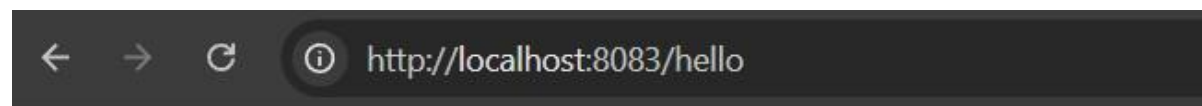
HelloController.java:

```
1 package com.cognizant.springlearn.controller;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController
9 public class HelloController {
10
11     private static final Logger LOGGER = LoggerFactory.getLogger(HelloController.class);
12
13     @GetMapping("/hello")
14     public String sayHello() {
15         LOGGER.info("START - sayHello()");
16         String response = "Hello World!!";
17         LOGGER.info("END - sayHello()");
18         return response;
19     }
20 }
21
```

Application.properties:

```
1 spring.application.name=spring-learn
2 server.port=8083
3
```

Output:



Hello World!!

## 2. REST - Country Web Service

Output:



### 1. What happens in the controller method?

When a GET request is made to /country, Spring maps it to `getCountryIndia()` because of the `@RequestMapping("/country")` annotation.

Inside the method:

A Spring XML application context is created and loads `date-format.xml`.

It retrieves the Country bean from XML (`<bean id="country" ...>`) — which contains code and name properties.

The Country object is returned.

### 2. How the bean is converted into JSON response?

Spring Boot automatically uses **Jackson** (a JSON serialization library) to convert Java objects to JSON format.

Behind the scenes:

You returned a Country object.

Spring uses `HttpMessageConverter`, specifically

`MappingJackson2HttpMessageConverter`, to:

Convert your Country object into JSON.

Set the appropriate Content-Type header: `application/json`.

### 3. In network tab of developer tools show the HTTP header details received To view this in Chrome:

1. Open your browser.
2. Go to `http://localhost:8083/country`.
3. Right-click anywhere → Inspect → Go to Network tab.
4. Refresh the page if needed.
5. Click on the request named /country.

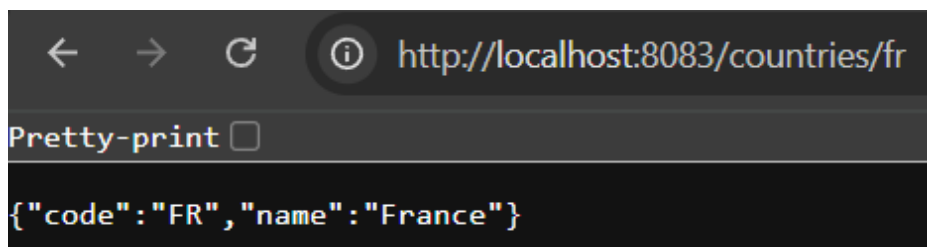
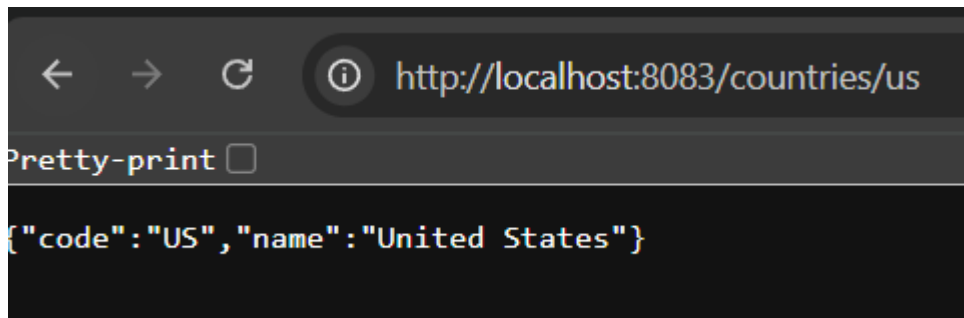
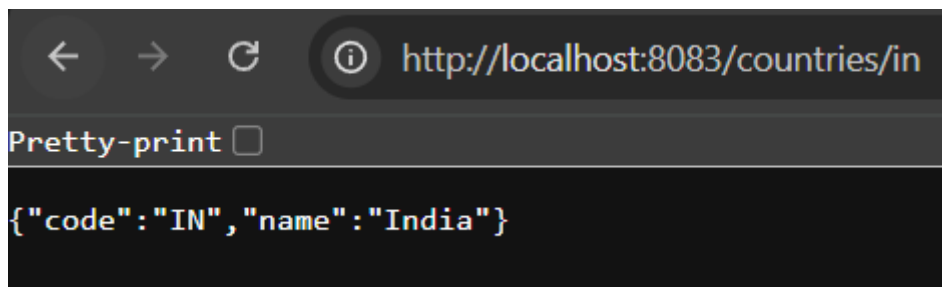
4. In postman click on "Headers" tab to view the HTTP header details received Steps:

1. Open Postman.
2. Make a GET request to:  

```
bash  
CopyEdit  
http://localhost:8083/country
```
3. Click Send.
4. Click the "Headers" tab in the response area (below the body).

### Get country based on country code

Output:



## 5. JWT-HANDSON:

Create authentication service that returns JWT :

