Student Name: SHIVENDRA KUMAR KUKKUDAPU

Registration No: 22072A04P3

Course/Batch: VNRVJIET (ECE)

HANDS ON 1: CREATE A SPRING WEB PROJECT USING MAVEN

Introduction:

This project is a simple Spring Boot web application built using Maven that allows users to manage books with basic CRUD operations (Create, Read, Update, Delete). It leverages Spring MVC, Thymeleaf for the frontend, and Spring Data JPA with a database for persistence.

Objective:

- To understand how to create a Spring Boot web application using Maven and configure dependencies using pom.xml.
- To implement a complete Book Management System with controller, service, and repository layers following MVC architecture.
- To create a user-friendly frontend using Thymeleaf for interacting with the backend via forms and dynamic data binding.

Implementation Breakdown:

SpringlearnApplication.java:

```
package com.cognizant.springlearn;

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);
}
}
```

Book.java:

```
package com.cognizant.springlearn;

import jakarta.persistence.*;

@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String author;
    private double price;

    public Book() {}

    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }
}
```

BookController.java:

```
package com.cognizant.springlearn;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
@Controller
```

```
public class BookController {

    @Autowired
    private BookService service;

    @GetMapping("/books")
    public String viewBooks(Model model) {
        model.addAttribute("books", service.getAllBooks());
        return "books";
    }

    @GetMapping("/books/add")
    public String showAddForm(Model model) {
        model.addAttribute("book", new Book());
        return "book-form";
    }

    @PostMapping("/books/save")
    public String save(@ModelAttribute("book") Book book) {
        service.saveBook(book);
        return "redirect:/books";
    }

    @GetMapping("/books/edit/{id}")
    public String edit(@PathVariable Long id, Model model) {
        model.addAttribute("book", service.getBookById(id));
        return "book-form";
    }

    @GetMapping("/books/delete/{id}")
    public String delete(@PathVariable Long id) {
        service.deleteBook(id);
        return "redirect:/books";
    }
}
```

BookRepository.java:

```
package com.cognizant.springlearn;

import org.springframework.data.jpa.repository.JpaRepository;
public interface BookRepository extends JpaRepository<Book, Long> {
}
```

BookService.java:

```
package com.cognizant.springlearn;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class BookService {

    private final BookRepository repository;
    @Autowired
    public BookService(BookRepository repository) {
        this.repository = repository;
    }

    public List<Book> getAllBooks() {
        return repository.findAll();
    }

    public void saveBook(Book book) {
        repository.save(book);
    }

    public Book getBookById(Long id) {
        return repository.findById(id).orElse(null);
    }

    public void deleteBook(Long id) {
        repository.deleteById(id);
    }
}
```

BookServiceTest.java:

```
package com.cognizant.springlearn;

import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;
```

```
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

public class BookServiceTest {

    @Test
    public void testGetAllBooks() {
        BookRepository mockRepo = Mockito.mock(BookRepository.class);
        List<Book> mockBooks = Arrays.asList(
                new Book("Book A", "Author A", 100.0),
                new Book("Book B", "Author B", 200.0)
        );
        when(mockRepo.findAll()).thenReturn(mockBooks);
        BookService service = new BookService(mockRepo);
        List<Book> books = service.getAllBooks();
        assertEquals(2, books.size());
        assertEquals("Book A", books.get(0).getTitle());
    }

    @Test
    public void testSaveBook() {
        BookRepository mockRepo = mock(BookRepository.class);
        BookService service = new BookService(mockRepo);

        Book book = new Book("Test Book", "Tester", 150.0);
        service.saveBook(book);

        verify(mockRepo, times(1)).save(book);
    }

    @Test
    public void testGetBookById() {
        BookRepository mockRepo = mock(BookRepository.class);
        Book book = new Book("Java Book", "Oracle", 299.99);
        book.setId(1L);
        when(mockRepo.findById(1L)).thenReturn(Optional.of(book));

        BookService service = new BookService(mockRepo);
        Book foundBook = service.getBookById(1L);

        assertNotNull(foundBook);
        assertEquals("Java Book", foundBook.getTitle());
    }
```

```
    @Test
    public void testDeleteBook() {
        BookRepository mockRepo = mock(BookRepository.class);
        BookService service = new BookService(mockRepo);

        service.deleteBook(1L);
        verify(mockRepo, times(1)).deleteById(1L);
    }
}
```

Book.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add/Edit Book</title>
    <link rel="stylesheet" th:href="@{/style.css}" />
</head>
<body>
<div class="container">
    <h1 th:text="${book.id == null} ? 'Add Book' : 'Edit Book'"></h1>
    <form th:action="@{/books/save}" th:object="${book}" method="post">
        <input type="hidden" th:field="*{id}" />
        <label>Title:</label>
        <input type="text" th:field="*{title}" required/><br/>
        <label>Author:</label>
        <input type="text" th:field="*{author}" required/><br/>
        <label>Price:</label>
        <input type="number" step="0.01" th:field="*{price}" required/><br/>
        <button type="submit" class="btn">Save</button>
    </form>
</div>
</body>
</html>
```

Book-form.html:

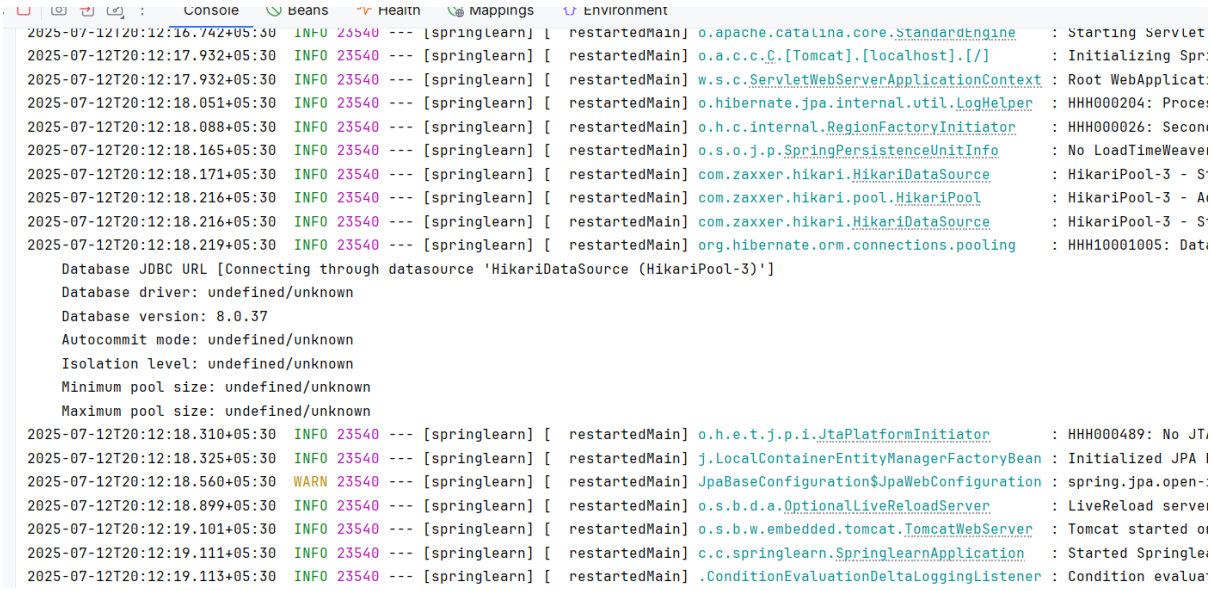
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Books</title>
    <link rel="stylesheet" th:href="@{/style.css}" />
</head>
<body>
```

```
    <div class="container">
        <h1>Book List</h1>
        <a class="btn" href="/books/add">Add New Book</a>
        <table>
<tr><th>ID</th><th>Title</th><th>Author</th><th>Price</th><th>Action</th></tr>
            <tr th:each="book : ${books}">
                <td th:text="${book.id}"></td>
                <td th:text="${book.title}"></td>
                <td th:text="${book.author}"></td>
                <td th:text="${book.price}"></td>
                <td>
                    <a th:href="@{/books/edit/{id}(id=${book.id})}">Edit</a> |
                    <a th:href="@{/books/delete/{id}(id=${book.id})}">Delete</a>
                </td>
            </tr>
        </table>
    </div>
    </body>
    </html>
```

Application.properties:

```
spring.application.name=springlearn
server.port=8080

spring.datasource.url=jdbc:mysql://localhost:3306/bookverse
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.thymeleaf.cache=false
```

Output:


Console
Beans
Health
Mappings
Environment
2025-07-12T20:12:16.742+05:30 INFO 23540 --- [springlearn] [restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet
2025-07-12T20:12:17.932+05:30 INFO 23540 --- [springlearn] [restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spr
2025-07-12T20:12:17.932+05:30 INFO 23540 --- [springlearn] [restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicat
2025-07-12T20:12:18.051+05:30 INFO 23540 --- [springlearn] [restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Proce
2025-07-12T20:12:18.088+05:30 INFO 23540 --- [springlearn] [restartedMain] o.h.c.internal.RegionFactoryInitiator : HHH000026: Secon
2025-07-12T20:12:18.165+05:30 INFO 23540 --- [springlearn] [restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeave
2025-07-12T20:12:18.171+05:30 INFO 23540 --- [springlearn] [restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-3 - S
2025-07-12T20:12:18.216+05:30 INFO 23540 --- [springlearn] [restartedMain] com.zaxxer.hikari.pool.HikariPool : HikariPool-3 - A
2025-07-12T20:12:18.216+05:30 INFO 23540 --- [springlearn] [restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-3 - S
2025-07-12T20:12:18.219+05:30 INFO 23540 --- [springlearn] [restartedMain] org.hibernate.orm.connections.pooling : HHH10001005: Data
Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-3)']
Database driver: undefined/unknown
Database version: 8.0.37
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-07-12T20:12:18.310+05:30 INFO 23540 --- [springlearn] [restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA
2025-07-12T20:12:18.325+05:30 INFO 23540 --- [springlearn] [restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA
2025-07-12T20:12:18.560+05:30 WARN 23540 --- [springlearn] [restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-
2025-07-12T20:12:18.899+05:30 INFO 23540 --- [springlearn] [restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server
2025-07-12T20:12:19.101+05:30 INFO 23540 --- [springlearn] [restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started o
2025-07-12T20:12:19.111+05:30 INFO 23540 --- [springlearn] [restartedMain] c.c.springlearn.SpringlearnApplication : Started Springle
2025-07-12T20:12:19.113+05:30 INFO 23540 --- [springlearn] [restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evalua
Add New Book
ID
TITLE
AUTHOR
PRICE
ACTION
1 Java Haritha 320.0 Edit | Delete
2 Wings of Fire A P J Abdul Kalam 299.0 Edit | Delete
3 The Alchemist Paulo Coelho 350.0 Edit | Delete
4 Harry Potter and the Sorcerer's Stone J.K. Rowling 499.0 Edit | Delete
5 The Monk Who Sold His Ferrari Robin Sharma 325.0 Edit | Delete
6 You Can Win Shiv Khera 275.0 Edit | Delete
7 Think and Grow Rich Napoleon Hill 399.0 Edit | Delete
8 Rich Dad Poor Dad Robert T. Kiyosaki 450.0 Edit | Delete
9 The Secret Rhonda Byrne 399.0 Edit | Delete
10 Life's Amazing Secrets Gaur Gopal Das 280.0 Edit | Delete

Add Book

Title:

Author:

Price:

0.0

Save

Edit Book

Title:

Java

Author:

Haritha

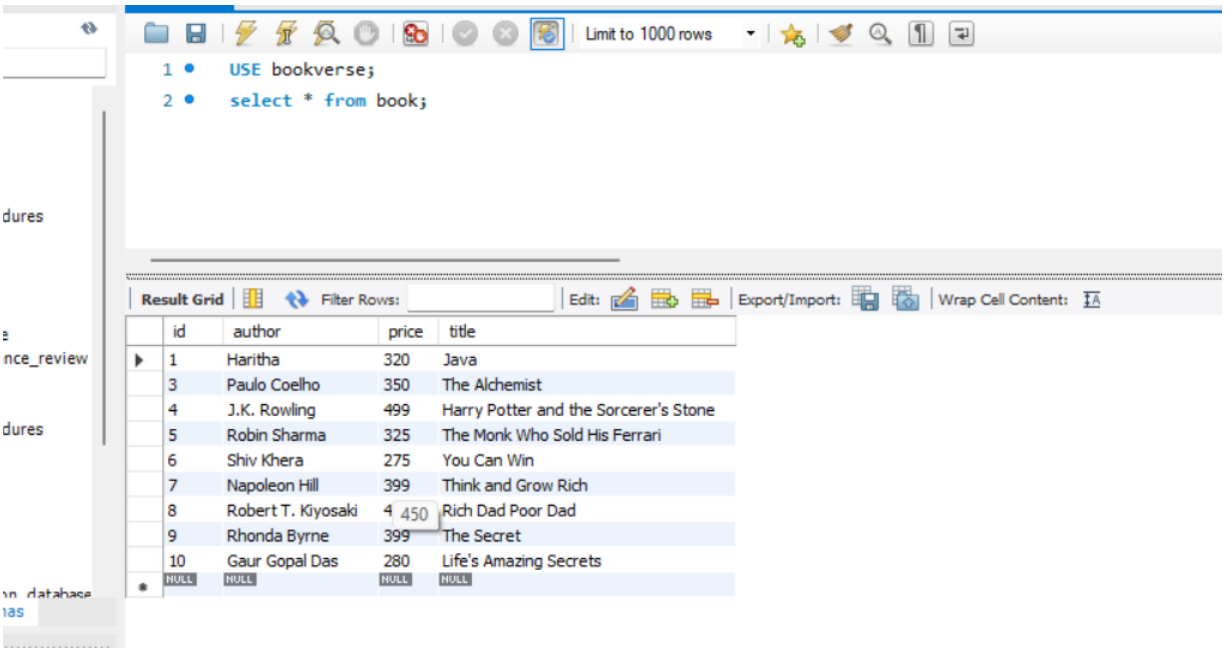
Price:

320.0

Save

Add New Book

ID	TITLE	AUTHOR	PRICE	ACTION
1	Java	Haritha	320.0	Edit | Delete
3	The Alchemist	Paulo Coelho	350.0	Edit | Delete
4	Harry Potter and the Sorcerer's Stone	J.K. Rowling	499.0	Edit | Delete
5	The Monk Who Sold His Ferrari	Robin Sharma	325.0	Edit | Delete
6	You Can Win	Shiv Khera	275.0	Edit | Delete
7	Think and Grow Rich	Napoleon Hill	399.0	Edit | Delete
8	Rich Dad Poor Dad	Robert T. Kiyosaki	450.0	Edit | Delete
9	The Secret	Rhonda Byrne	399.0	Edit | Delete
10	Life's Amazing Secrets	Gaur Gopal Das	280.0	Edit | Delete



Conclusion:

This project demonstrates the integration of key Spring Boot components to build a functional web application. It serves as a practical example of applying backend and frontend concepts together in a real-world scenario using Java and Spring.

HANDS ON 6:SpringCore – Load Country from Spring Configuration XML

Introduction:

This Spring Core application demonstrates how to use **Spring's XML-based configuration** to define and inject a list of country beans. The goal is to simulate retrieving and displaying country data, such as for an airline website, using traditional Spring dependency injection and logging mechanisms.

Objective:

- To create and configure multiple Spring beans representing countries using the country.xml file.
- To load a list of country objects into an ArrayList bean using XML <list> and <ref> tags.
- To programmatically retrieve and display the list of countries using a Java class (SpringlearnApplication) with debug-level logging via SLF4J.

Implementation Breakdown:

SpringlearnApplication.java:

```
package com.cognizant.springlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.List;

public class SpringlearnApplication {

private static final Logger LOGGER =
LoggerFactory.getLogger(SpringlearnApplication.class);


public static void main(String[] args) {

        LOGGER.info("START main()");

        System.out.println("Main method started");
```

```
        try {
                ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("country.xml");

                List<Country> countryList = (List<Country>)
context.getBean("countryList");
                System.out.println("Loaded countries: " + countryList.size());

                for (Country country : countryList) {
                        System.out.println("Country: " + country); // fallback print
                        LOGGER.debug("Country: {}", country);
                }

                context.close();
        } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
                e.printStackTrace();
        }

        LOGGER.info("END main()");
}
}
```

Country.java:

```
package com.cognizant.springlearn;
public class Country {
   private String code;
   private String name;

   public Country() {
```

```
    }

    public Country(String code, String name) {
        this.code = code;
        this.name = name;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Country{" + "code='" + code + '\'' + ", name='" + name + '\'' + '}';
    }
}
```

Country.xml:

```
<?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
         http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="in" class="com.cognizant.springlearn.Country">
      <property name="code" value="IN"/>
      <property name="name" value="India"/>
   </bean>

   <bean id="us" class="com.cognizant.springlearn.Country">
      <property name="code" value="US"/>
      <property name="name" value="United States"/>
   </bean>

   <bean id="de" class="com.cognizant.springlearn.Country">
      <property name="code" value="DE"/>
      <property name="name" value="Germany"/>
   </bean>

   <bean id="jp" class="com.cognizant.springlearn.Country">
      <property name="code" value="JP"/>
      <property name="name" value="Japan"/>
   </bean>

   <bean id="fr" class="com.cognizant.springlearn.Country">
```

```
    <property name="code" value="FR"/>
    <property name="name" value="France"/>
</bean>

<bean id="uk" class="com.cognizant.springlearn.Country">
    <property name="code" value="UK"/>
    <property name="name" value="United Kingdom"/>
</bean>

<bean id="au" class="com.cognizant.springlearn.Country">
    <property name="code" value="AU"/>
    <property name="name" value="Australia"/>
</bean>

<bean id="ca" class="com.cognizant.springlearn.Country">
    <property name="code" value="CA"/>
    <property name="name" value="Canada"/>
</bean>

<bean id="cn" class="com.cognizant.springlearn.Country">
    <property name="code" value="CN"/>
    <property name="name" value="China"/>
</bean>

<bean id="br" class="com.cognizant.springlearn.Country">
    <property name="code" value="BR"/>
    <property name="name" value="Brazil"/>
</bean>
<bean id="countryList" class="java.util.ArrayList">
```

```
        <constructor-arg>
            <list>
                <ref bean="in"/>
                <ref bean="us"/>
                <ref bean="de"/>
                <ref bean="jp"/>
                <ref bean="fr"/>
                <ref bean="uk"/>
                <ref bean="au"/>
                <ref bean="ca"/>
                <ref bean="cn"/>
                <ref bean="br"/>
            </list>
        </constructor-arg>
    </bean>
</beans>
```

Application.properties:

```
org.slf4j.simpleLogger.defaultLogLevel=debug
org.slf4j.simpleLogger.showDateTime=true
org.slf4j.simpleLogger.showThreadName=true
org.slf4j.simpleLogger.dateTimeFormat=yyyy-MM-dd HH:mm:ss
```

Output:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
[main] INFO com.cognizant.springlearn.SpringlearnApplication - START main()
Main method started
Loaded countries: 10
Country: Country{code='IN', name='India'}
Country: Country{code='US', name='United States'}
Country: Country{code='DE', name='Germany'}
Country: Country{code='JP', name='Japan'}
Country: Country{code='FR', name='France'}
Country: Country{code='UK', name='United Kingdom'}
Country: Country{code='AU', name='Australia'}
Country: Country{code='CA', name='Canada'}
Country: Country{code='CN', name='China'}
Country: Country{code='BR', name='Brazil'}
[main] INFO com.cognizant.springlearn.SpringlearnApplication - END main()

Process finished with exit code 0
```

Conclusion:

This hands-on project successfully demonstrates how to define complex object graphs (like a list of countries) in Spring using XML configuration. It provides practical experience in bean declaration, collection injection, and dependency management, forming a foundational understanding of Spring's core container and its configuration styles.

Hello World RESTful Web Service

Introduction:

This project is a simple Spring Boot application that demonstrates the creation of a RESTful web service using Spring Web. The service provides a basic endpoint (/hello) that returns the message "Hello World!!" and logs its execution using SLF4J.

Objective:

- To build a RESTful web service using Spring Boot and expose a GET endpoint.
- To apply the @RestController and @GetMapping annotations for handling web requests.
- To implement basic logging using SLF4J to track method entry and exit points.

Implementation:

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) {

        SpringApplication.run(SpringLearnApplication.class, args);

}

}
```

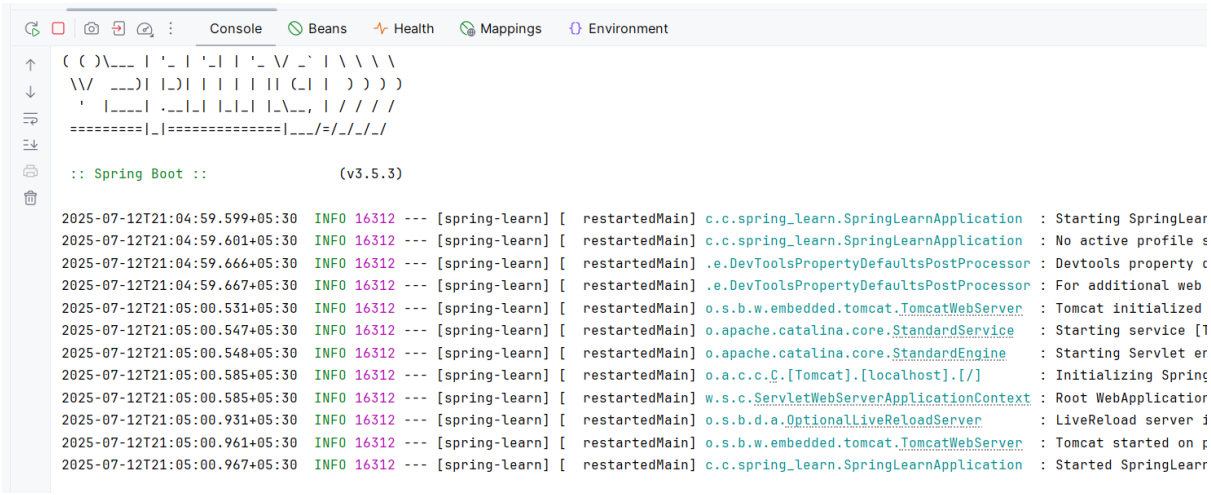
HelloController.java:

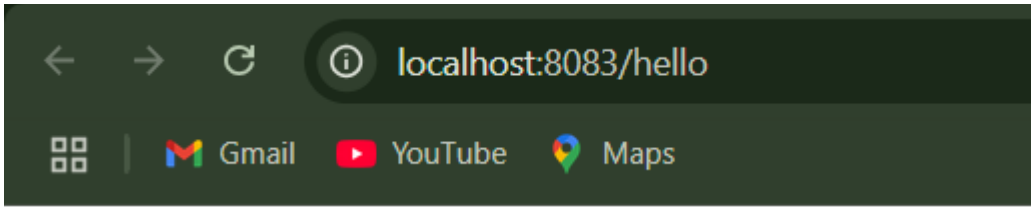
```
package com.cognizant.spring_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HelloController {

    private static final Logger LOGGER =
LoggerFactory.getLogger(HelloController.class);

    @GetMapping("/hello")

    public String sayHello() {

        LOGGER.info("START sayHello()");

        String message = "Hello World!!";

        LOGGER.info("END sayHello()");

        return message;

    }

}
```

Application.properties:

```
spring.application.name=spring-learn
server.port=8083
```

Output:





Hello World!!

Conclusion:

This hands-on exercise introduces the core concepts of Spring Boot REST development. By successfully building and running a /hello endpoint, it provides a foundation for developing more complex REST APIs and familiarizes developers with essential annotations and logging practices.

REST – Country Web Service

Introduction:

This project is a Spring Boot RESTful web service that returns a list of countries in JSON format. The application uses a simple GET API endpoint to provide data representing country codes and names, with logging to trace method execution.

Objective:

- To build a RESTful endpoint /countries that returns a hardcoded list of 20 countries.
- To define a Country model class for representing country data with code and name attributes.
- To implement logging using SLF4J to track the execution flow of the API controller method.

Implementation:

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) {

		SpringApplication.run(SpringLearnApplication.class, args);

	}

}
```

Country.java:

```
package com.cognizant.spring_learn;

public class Country {
    private String code;
    private String name;


    public Country() {}
    public Country(String code, String name) {
        this.code = code;
        this.name = name;
    }
    public String getCode() {
        return code;
    }
    public void setCode(String code) {
        this.code = code;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

CountryController.java:

```
package com.cognizant.spring_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;


import java.util.Arrays;

import java.util.List;


@RestController

public class CountryController {


   private static final Logger LOGGER = LoggerFactory.getLogger(CountryController.class);


   @GetMapping("/countries")

   public List<Country> getAllCountries() {

      LOGGER.info("START getAllCountries()");


      List<Country> countries = Arrays.asList(

            new Country("IN", "India"),

            new Country("US", "United States"),

            new Country("UK", "United Kingdom"),
```

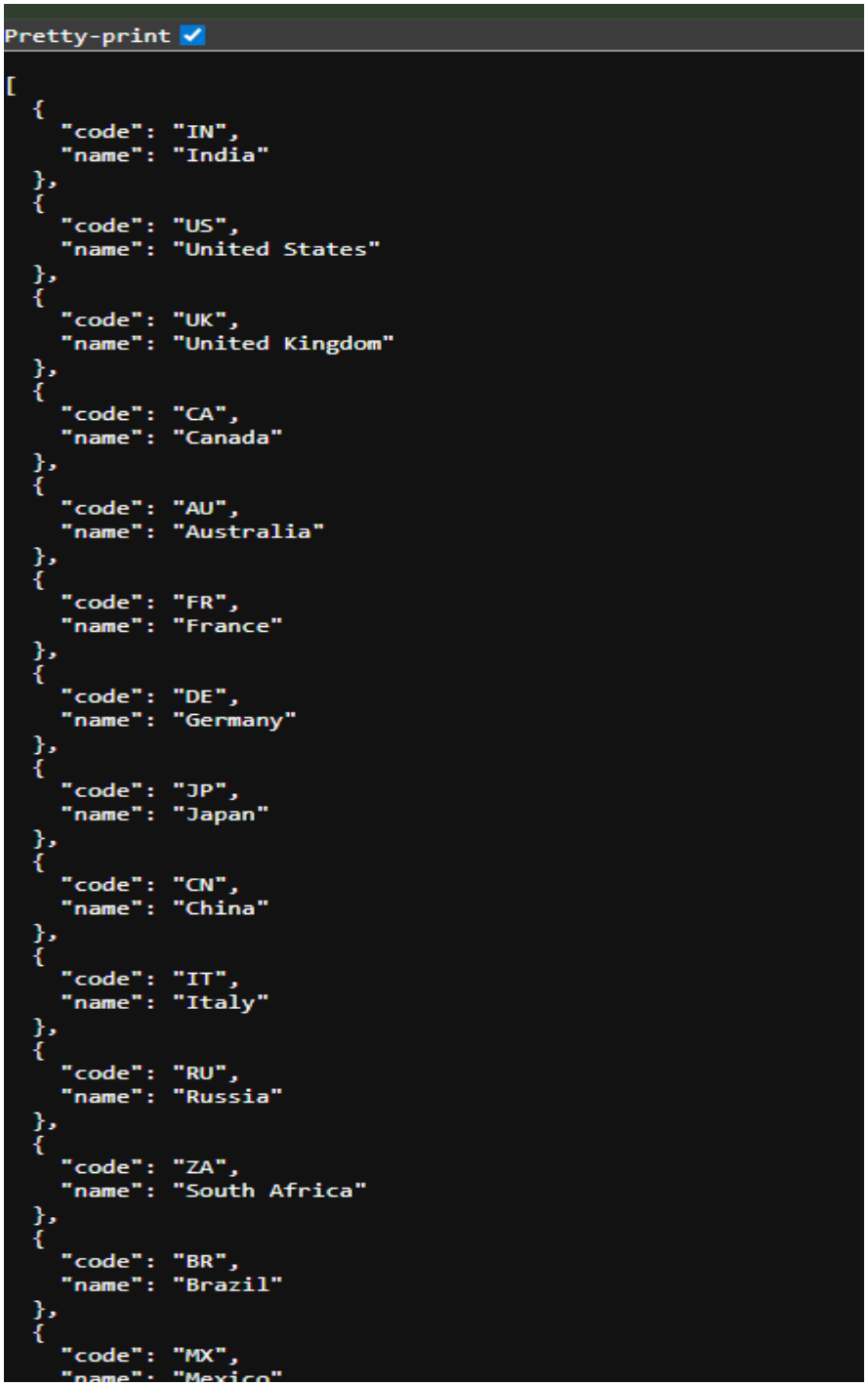
```
            new Country("CA", "Canada"),
            new Country("AU", "Australia"),
            new Country("FR", "France"),
            new Country("DE", "Germany"),
            new Country("JP", "Japan"),
            new Country("CN", "China"),
            new Country("IT", "Italy"),
            new Country("RU", "Russia"),
            new Country("ZA", "South Africa"),
            new Country("BR", "Brazil"),
            new Country("MX", "Mexico"),
            new Country("KR", "South Korea"),
            new Country("AE", "UAE"),
            new Country("SG", "Singapore"),
            new Country("NZ", "New Zealand"),
            new Country("ES", "Spain"),
            new Country("SE", "Sweden")
        );

        LOGGER.info("END getAllCountries()");
        return countries;
    }
}
```

Application.properties:

spring.application.name=spring-learn

server.port=8083

Output:



```
    "name": "Italy"
  },
  {
    "code": "RU",
    "name": "Russia"
  },
  {
    "code": "ZA",
    "name": "South Africa"
  },
  {
    "code": "BR",
    "name": "Brazil"
  },
  {
    "code": "MX",
    "name": "Mexico"
  },
  {
    "code": "KR",
    "name": "South Korea"
  },
  {
    "code": "AE",
    "name": "UAE"
  },
  {
    "code": "SG",
    "name": "Singapore"
  },
  {
    "code": "NZ",
    "name": "New Zealand"
  },
  {
    "code": "ES",
    "name": "Spain"
  },
  {
    "code": "SE",
    "name": "Sweden"
  }
]
```


Conclusion:

This hands-on project provides a clear understanding of how to create and expose REST APIs using Spring Boot. By successfully returning a structured list of country objects, it serves as a foundation for building scalable, data-driven microservices with proper logging and JSON response handling.

REST – Get Country Based on Country Code

Introduction:

This Spring Boot RESTful service allows users to retrieve country details based on a given country code. Using Spring's XML configuration and dependency injection, the application loads a predefined list of countries and returns the matching country when requested via an HTTP GET endpoint.

Objective:

- To implement a RESTful endpoint /country/{code} that returns the details of a specific country.
- To read a list of countries from an XML configuration file (country.xml) using Spring's IoC container.
- To enable case-insensitive lookup of country codes using Java Stream API for clean and efficient filtering.

Implementation:

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) {

	SpringApplication.run(SpringLearnApplication.class, args);

}

}
```

Country.java:

```
package com.cognizant.spring_learn;

public class Country {
    private String code;
    private String name;


    public Country() {}
    public Country(String code, String name) {
        this.code = code;
        this.name = name;
    }
    public String getCode() {
        return code;
    }
    public void setCode(String code) {
        this.code = code;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

CountryService.java:

```
package com.cognizant.spring_learn;


import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class CountryService {


    public Country getCountry(String code) {
        ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("country.xml");


        List<Country> countryList = (List<Country>) context.getBean("countryList");


        Country country = countryList.stream()
                .filter(c -> c.getCode().equalsIgnoreCase(code))
                .findFirst()
                .orElse(null);


        context.close();
        return country;
    }
}
```

CountryController.java:

```
package com.cognizant.spring_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.*;


@RestController

public class CountryController {


    private static final Logger LOGGER =
LoggerFactory.getLogger(CountryController.class);


    @Autowired

    private CountryService countryService;


    @GetMapping("/country/{code}")

    public Country getCountry(@PathVariable String code) {

        LOGGER.info("START getCountry() for code: {}", code);

        Country country = countryService.getCountry(code);

        LOGGER.info("END getCountry()");

        return country;

    }

}
```

Country.xml:

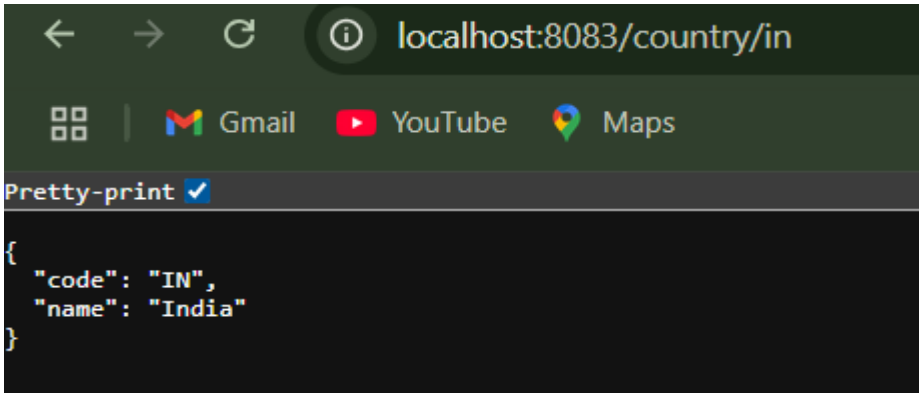
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
   <bean id="countryList" class="java.util.ArrayList">
      <constructor-arg>
         <list>
            <ref bean="in"/>
            <ref bean="us"/>
            <ref bean="uk"/>
            <ref bean="de"/>
         </list>
      </constructor-arg>
   </bean>
   <bean id="in" class="com.cognizant.spring_learn.Country">
      <property name="code" value="IN"/>
      <property name="name" value="India"/>
   </bean>
   <bean id="us" class="com.cognizant.spring_learn.Country">
      <property name="code" value="US"/>
      <property name="name" value="United States"/>
```

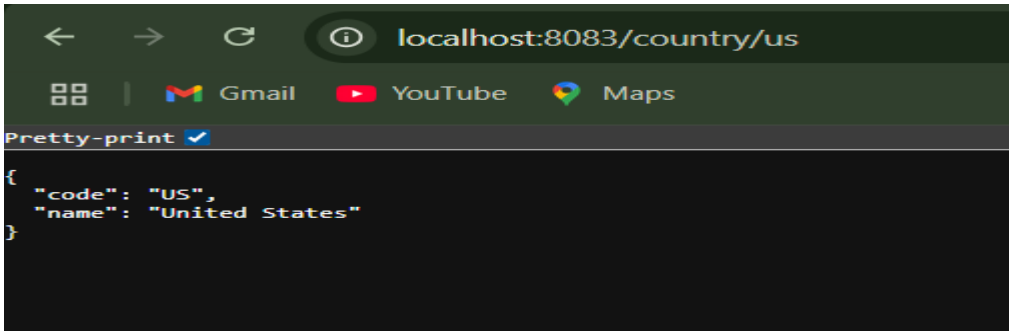
```
    </bean>
    <bean id="uk" class="com.cognizant.spring_learn.Country">
        <property name="code" value="UK"/>
        <property name="name" value="United Kingdom"/>
    </bean>
    <bean id="de" class="com.cognizant.spring_learn.Country">
        <property name="code" value="DE"/>
        <property name="name" value="Germany"/>
    </bean>
</beans>
```

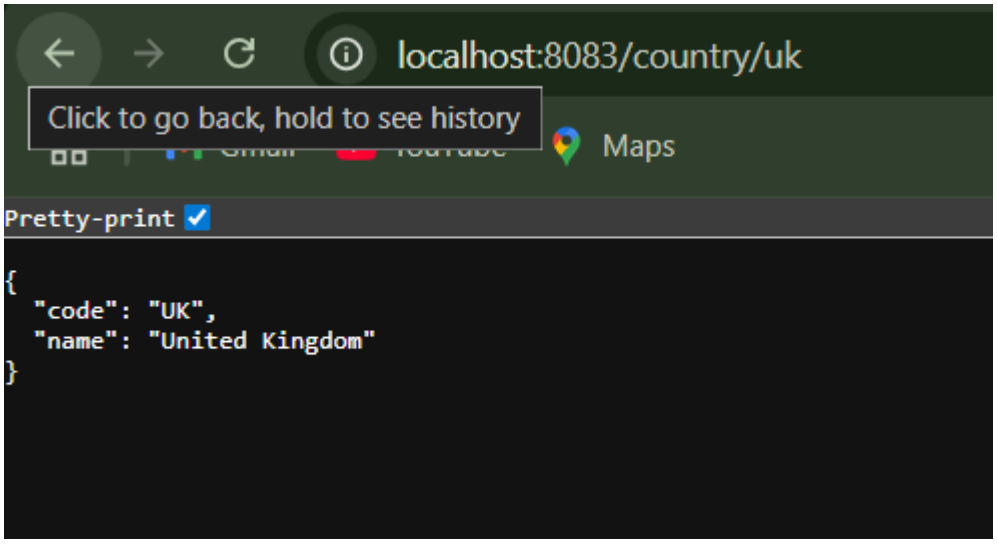
Application.properties:

```
spring.application.name=spring-learn
server.port=8083
```

Output:







Conclusion:

This exercise demonstrates how to integrate Spring XML configuration with a REST controller to serve dynamic responses. It also highlights the use of @PathVariable, service layering, and Spring dependency injection to create a clean, modular RESTful service. The approach is scalable and can be easily extended to support more countries or dynamic data sources like databases.