**Hands on 1**

**Create a Spring Web Project using Maven**   
  
Follow steps below to create a project: 

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.

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

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. SpringLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.

spring‑learn (Project Structre)

├─ src

│ ├─ main

│ │ ├─ java

│ │ │ └─ com.cognizant.springlearn

│ │ │ └─ SpringLearnApplication.java

│ │ └─ resources

│ │ └─ application.properties

│ └─ test

│ └─ java

│ └─ com.cognizant.springlearn

│ └─ SpringLearnApplicationTests.java

├─ pom.xml

└─ .mvn/ mvnw mvnw.cmd

SpringLearnApplication.java

package com.cognizant.springlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

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

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

LOGGER.info("Inside main of SpringLearnApplication");

}

}

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>

<packaging>war</packaging>

<name>spring\_learn</name>

<description>Demo project for Spring Web</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>21</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-tomcat</artifactId>

<scope>provided</scope>

</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>

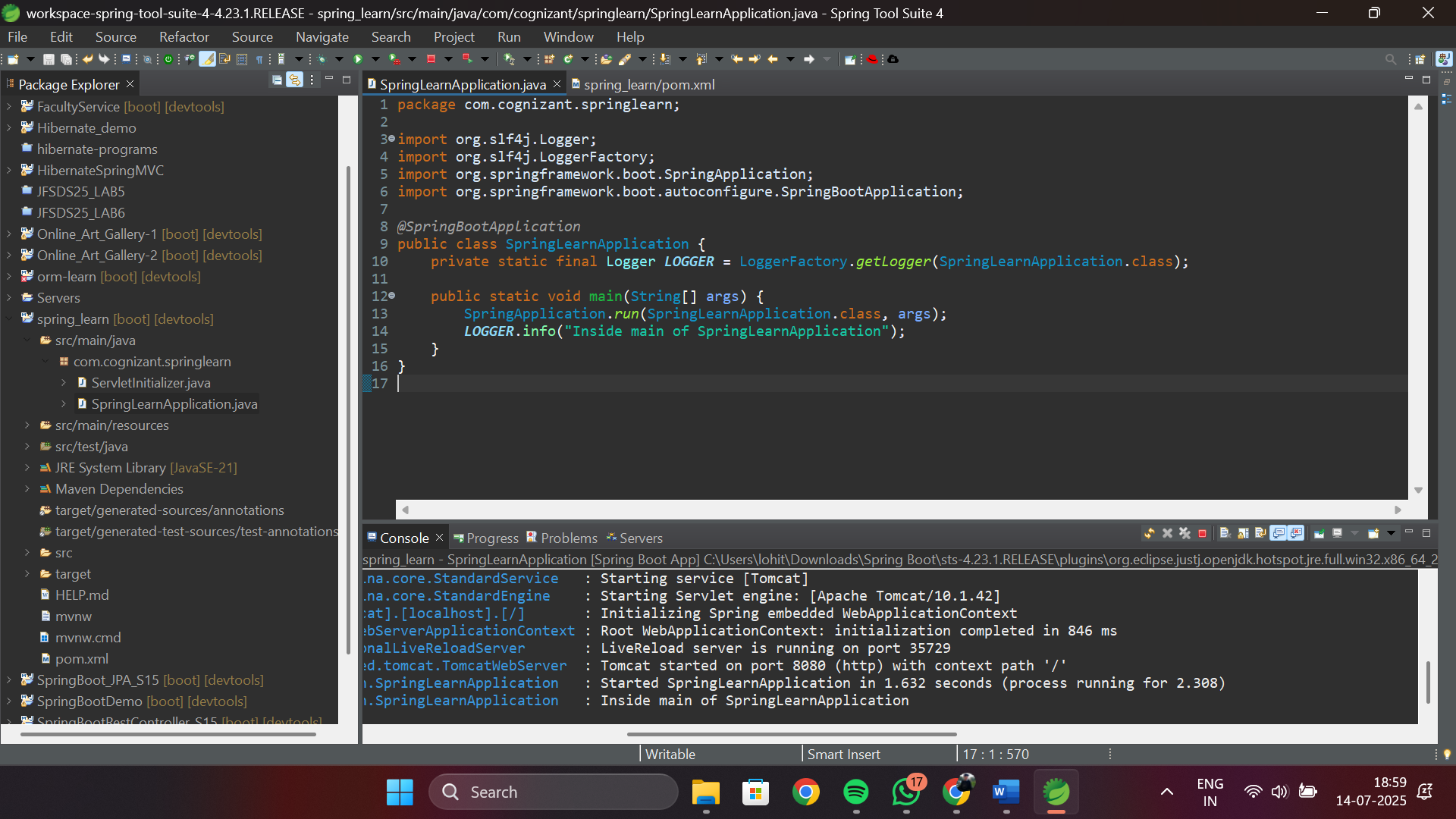
<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>



**Hands on 2**

**Spring Core – Load Country from Spring Configuration XML**   
  
An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country. 

|  |  |
| --- | --- |
| **Code** | **Name** |
| US | United States |
| DE | Germany |
| IN | India |
| JP | Japan |

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details.  
  
Steps to implement

* Pick any one of your choice country to configure in Spring XML configuration named country.xml.
* Create a bean tag in spring configuration for country and set the property and values

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

        <property name="code" value="IN" />

        <property name="name" value="India" />

    </bean>

* Create Country class with following aspects:
  + Instance variables for code and name
  + Implement empty parameter constructor with inclusion of debug log within the constructor with log message as “Inside Country Constructor.”
  + Generate getters and setters with inclusion of debug with relevant message within each setter and getter method.
  + Generate toString() method
* Create a method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country details. ClassPathXmlApplicationContext, ApplicationContext and context.getBean(“beanId”, Country.class). Refer sample code for displayCountry() method below.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

Country country = (Country) context.getBean("country", Country.class);

LOGGER.debug("Country : {}", country.toString());

* Invoke displayCountry() method in main() method of SpringLearnApplication.java.
* Execute main() method and check the logs to find out which constructors and methods were invoked.

SME to provide more detailing about the following aspects:

* bean tag, id attribute, class attribute, property tag, name attribute, value attribute
* ApplicationContext, ClassPathXmlApplicationContext
* What exactly happens when context.getBean() is invoked

Add an XML config file

<?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

https://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Define one Country bean -->

<bean id="country" class="com.cognizant.springlearn.model.Country">

<property name="code" value="IN"/>

<property name="name" value="India"/>

</bean>

</beans>

Create the Country class

package com.cognizant.springlearn.model;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Country {

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

private String code;

private String name;

// No-arg constructor

public Country() {

LOGGER.debug("Inside Country constructor");

}

public String getCode() {

LOGGER.debug("In getCode()");

return code;

}

public void setCode(String code) {

LOGGER.debug("In setCode({})", code);

this.code = code;

}

public String getName() {

LOGGER.debug("In getName()");

return name;

}

public void setName(String name) {

LOGGER.debug("In setName({})", name);

this.name = name;

}

@Override

public String toString() {

return String.format("Country{code='%s', name='%s'}", code, name);

}

}

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.cognizant.springlearn.model.Country;

// …

public class SpringLearnApplication {

// existing logger & main()

private static void displayCountry() {

// 1) Load the XML application context

ApplicationContext context =

new ClassPathXmlApplicationContext("country.xml");

// 2) Retrieve the country bean

Country country = context.getBean("country", Country.class);

// 3) Log its details

LOGGER.debug("Country : {}", country);

}

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

LOGGER.info("Inside main of SpringLearnApplication");

// invoke our new method

displayCountry();

}

}

A computer screen shot of a program

AI-generated content may be incorrect.

HandsOn 3

**Hello World RESTful Web Service**   
  
Write a REST service in the spring learn application created earlier, that returns the text "Hello World!!" using Spring Web Framework. Refer details below:  
  
**Method:** GET  
**URL:** /hello  
**Controller:** com.cognizant.spring-learn.controller.HelloController  
**Method Signature:** public String sayHello()  
**Method Implementation:** return hard coded string "Hello World!!"  
**Sample Request**: http://localhost:8083/hello  
**Sample Response:** Hello World!!   
  
**IMPORTANT NOTE**: Don't forget to include start and end log in the sayHello() method.  
  
Try the URL http://localhost:8083/hello in both chrome browser and postman.  
  
SME to explain the following aspects:

* In network tab of developer tools show the HTTP header details received
* In postman click on "Headers" tab to view the HTTP header details received

**Change the Server Port to 8083**

By default Spring Boot runs on 8080, but your sample uses 8083. In src/main/resources/application.properties add:

properties

CopyEdit

server.port=8083

package com.cognizant.springlearn.controller;

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("sayHello() - start");

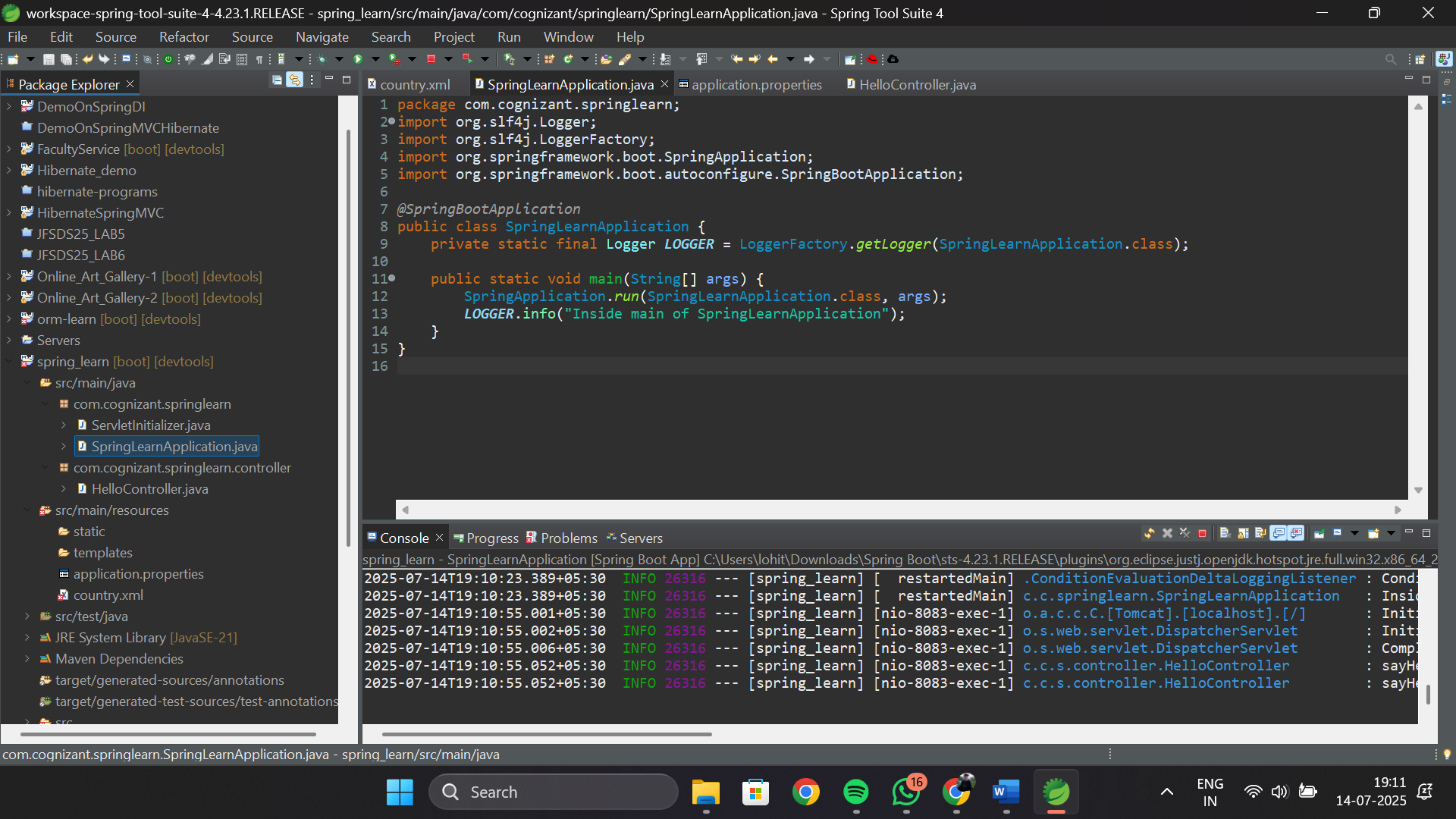
String response = "Hello World!!";

LOGGER.info("sayHello() - end");

return response;

}

}



A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

HandOn 4

**REST - Country Web Service**   
  
Write a REST service that returns India country details in the earlier created spring learn application.  
  
**URL**: /country  
**Controller**: com.cognizant.spring-learn.controller.CountryController  
**Method Annotation**: @RequestMapping  
**Method Name**: getCountryIndia()  
**Method Implementation**: Load India bean from spring xml configuration and return  
**Sample Request**: http://localhost:8083/country  
**Sample Response**:

{

  "code": "IN",

  "name": "India"

}

SME to explain the following aspects:

* What happens in the controller method?
* How the bean is converted into JSON reponse?
* In network tab of developer tools show the HTTP header details received
* In postman click on "Headers" tab to view the HTTP header details received

package com.cognizant.springlearn.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.cognizant.springlearn.model.Country;

@RestController

public class CountryController {

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

@RequestMapping(value = "/country", method = RequestMethod.GET)

public Country getCountryIndia() {

LOGGER.info("getCountryIndia() - start");

// Load the XML context and retrieve the 'country' bean

ApplicationContext context =

new ClassPathXmlApplicationContext("country.xml");

Country india = context.getBean("country", Country.class);

LOGGER.info("getCountryIndia() - end");

return india; // Spring MVC + Jackson will convert this to JSON

}

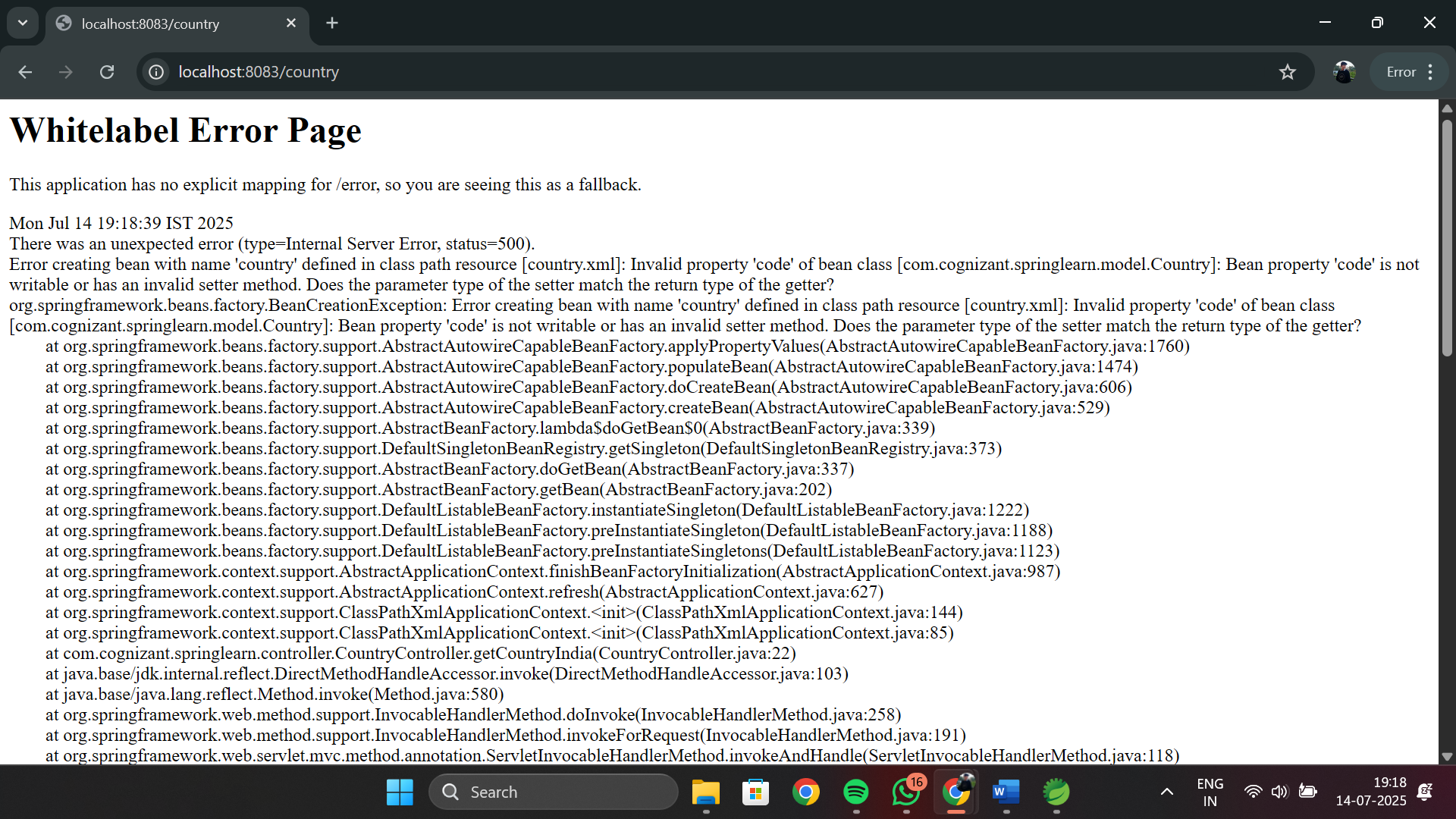
}

{

"code": "IN",

"name": "India"

}



HandsOn 5

**REST - Get country based on country code**   
  
Write a REST service that returns a specific country based on country code. The country code should be case insensitive.  
  
**Controller**: com.cognizant.spring-learn.controller.CountryController  
**Method Annotation:** @GetMapping("/countries/{code}")  
**Method Name**: getCountry(String code)  
**Method Implemetation**: Invoke countryService.getCountry(code)   
**Service Method:**com.cognizant.spring-learn.service.CountryService.getCountry(String code)  
  
**Service Method Implementation**:

* Get the country code using @PathVariable
* Get country list from country.xml
* Iterate through the country list
* Make a case insensitive matching of country code and return the country.
* Lambda expression can also be used instead of iterating the country list

**Sample Request**: http://localhost:8083/country/in  
  
**Sample Response**:

{

  "code": "IN",

  "name": "India"

}

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:util="http://www.springframework.org/schema/util"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

https://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/util

https://www.springframework.org/schema/util/spring-util.xsd">

<!-- Four countries -->

<bean id="countryUS" class="com.cognizant.countryservice.model.Country">

<property name="code" value="US"/>

<property name="name" value="United States"/>

</bean>

<bean id="countryDE" class="com.cognizant.countryservice.model.Country">

<property name="code" value="DE"/>

<property name="name" value="Germany"/>

</bean>

<bean id="countryIN" class="com.cognizant.countryservice.model.Country">

<property name="code" value="IN"/>

<property name="name" value="India"/>

</bean>

<bean id="countryJP" class="com.cognizant.countryservice.model.Country">

<property name="code" value="JP"/>

<property name="name" value="Japan"/>

</bean>

<!-- Aggregate into a list -->

<util:list id="countryList" value-type="com.cognizant.countryservice.model.Country">

<ref bean="countryUS"/>

<ref bean="countryDE"/>

<ref bean="countryIN"/>

<ref bean="countryJP"/>

</util:list>

</beans>

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*https://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/util*

*https://www.springframework.org/schema/util/spring-util.xsd"*>

<!-- Four countries -->

<**bean** id=*"countryUS"* class=*"com.cognizant.countryservice.model.Country"*>

<**property** name=*"code"* value=*"US"*/>

<**property** name=*"name"* value=*"United States"*/>

</**bean**>

<**bean** id=*"countryDE"* class=*"com.cognizant.countryservice.model.Country"*>

<**property** name=*"code"* value=*"DE"*/>

<**property** name=*"name"* value=*"Germany"*/>

</**bean**>

<**bean** id=*"countryIN"* class=*"com.cognizant.countryservice.model.Country"*>

<**property** name=*"code"* value=*"IN"*/>

<**property** name=*"name"* value=*"India"*/>

</**bean**>

<**bean** id=*"countryJP"* class=*"com.cognizant.countryservice.model.Country"*>

<**property** name=*"code"* value=*"JP"*/>

<**property** name=*"name"* value=*"Japan"*/>

</**bean**>

<!-- Aggregate into a list -->

<**util:list** id=*"countryList"* value-type=*"com.cognizant.countryservice.model.Country"*>

<**ref** bean=*"countryUS"*/>

<**ref** bean=*"countryDE"*/>

<**ref** bean=*"countryIN"*/>

<**ref** bean=*"countryJP"*/>

</**util:list**>

</**beans**>

package com.cognizant.countryservice.service;

import java.util.List;

import java.util.Optional;

import javax.annotation.PostConstruct;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.stereotype.Service;

import com.cognizant.countryservice.model.Country;

@Service

public class CountryService {

private List<Country> countries;

@PostConstruct

public void init() {

ApplicationContext ctx = new ClassPathXmlApplicationContext("country.xml");

//noinspection unchecked

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

}

/\*\*

\* Finds a country by ISO code (case‑insensitive). Returns null if not found.

\*/

public Country getCountry(String code) {

Optional<Country> match = countries.stream()

.filter(c -> c.getCode().equalsIgnoreCase(code))

.findFirst();

return match.orElse(null);

}

}

package com.cognizant.countryservice.controller;

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

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

import com.cognizant.countryservice.model.Country;

import com.cognizant.countryservice.service.CountryService;

@RestController

public class CountryController {

@Autowired

private CountryService countryService;

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

public Country getCountry(@PathVariable String code) {

return countryService.getCountry(code);

}

}

country‑service

├─ src

│ ├─ main

│ │ ├─ java

│ │ │ └─ com.cognizant.countryservice

│ │ │ ├─ CountryServiceApplication.java ← @SpringBootApplication

│ │ │ ├─ controller

│ │ │ │ └─ CountryController.java

│ │ │ ├─ model

│ │ │ │ └─ Country.java

│ │ │ └─ service

│ │ │ └─ CountryService.java

│ │ └─ resources

│ │ ├─ application.properties

│ │ └─ country.xml

│ └─ test

│ └─ java

│ └─ com.cognizant.countryservice

│ └─ CountryServiceApplicationTests.java

└─ pom.xml

HandsOn 6

**Create authentication service that returns JWT**   
  
As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.  
  
**Request**

curl -s -u user:pwd http://localhost:8090/authenticate

**Response**

{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}

This can be incorporated as three major steps:

* Create authentication controller and configure it in SecurityConfig
* Read Authorization header and decode the username and password
* Generate token based on the user retrieved in the previous step

Let incorporate the above as separate hands on exercises.

**Add Dependencies**

In your **pom.xml**, include:

xml

CopyEdit

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

package com.cognizant.springlearn.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.\*;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

// No user store—allow all; we'll manually authenticate in the controller

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable()

.authorizeRequests()

.antMatchers("/authenticate").permitAll()

.anyRequest().authenticated()

.and()

.httpBasic() // enable Basic auth for /authenticate

.and()

.sessionManagement().disable();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

package com.cognizant.springlearn.util;

import io.jsonwebtoken.\*;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

import java.security.Key;

import java.util.Date;

@Component

public class JwtUtil {

private Key key;

private final long validityMs = 15 \* 60 \* 1000; // 15 minutes

@PostConstruct

public void init() {

// Generate a random HS256 key; in prod load from config

this.key = Keys.secretKeyFor(SignatureAlgorithm.HS256);

}

public String generateToken(String username) {

Date now = new Date();

Date expiry = new Date(now.getTime() + validityMs);

return Jwts.builder()

.setSubject(username)

.setIssuedAt(now)

.setExpiration(expiry)

.signWith(key)

.compact();

}

// (optional) methods to validate and parse tokens...

}

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.util.JwtUtil;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.\*;

import org.springframework.http.ResponseEntity;

import org.springframework.security.crypto.password.PasswordEncoder;

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

import java.nio.charset.StandardCharsets;

import java.util.Base64;

@RestController

public class AuthController {

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

@Autowired

private JwtUtil jwtUtil;

@Autowired

private PasswordEncoder passwordEncoder;

// In-memory user for demo: username=user, password=encoded(pwd)

private final String demoUser = "user";

private final String demoPwdHash = passwordEncoder().encode("pwd");

// Expose /authenticate on port 8090

@GetMapping("/authenticate")

public ResponseEntity<?> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("authenticate() - start");

// 1) Extract Base64 part

if (authHeader == null || !authHeader.startsWith("Basic ")) {

return ResponseEntity.status(401).build();

}

String base64Cred = authHeader.substring("Basic ".length());

byte[] decoded = Base64.getDecoder().decode(base64Cred);

String[] parts = new String(decoded, StandardCharsets.UTF\_8).split(":", 2);

String username = parts[0], password = parts[1];

// 2) Validate credentials (demo only)

if (!demoUser.equals(username) || !passwordEncoder.matches(password, demoPwdHash)) {

return ResponseEntity.status(401).build();

}

// 3) Generate JWT

String token = jwtUtil.generateToken(username);

LOGGER.info("authenticate() - end");

return ResponseEntity.ok().body(Map.of("token", token));

}

// Bean method for PasswordEncoder injection

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

curl -i -u user:pwd <http://localhost:8090/authenticate>

Authorization: Basic dXNlcjpwd2Q=

HTTP/1.1 200 OK

Content-Type: application/json

{"token":"<your.jwt.here>"}