1. **Create a Spring Web Project using Maven**

[**spring-learn.zip**](spring-learn.zip)

**Output:**

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v3.5.3)

2025-07-13T09:01:52.336+05:30 INFO 23356 --- [spring-learn] [ restartedMain] c.c.springlearn.SpringLearnApplication : Starting SpringLearnApplication using Java 21.0.7 with PID 23356 (D:\Eclipse\SpringWorkspace\spring-learn\target\classes started by 91891 in D:\Eclipse\SpringWorkspace\spring-learn)

2025-07-13T09:01:52.340+05:30 INFO 23356 --- [spring-learn] [ restartedMain] c.c.springlearn.SpringLearnApplication : No active profile set, falling back to 1 default profile: "default"

2025-07-13T09:01:52.407+05:30 INFO 23356 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable

2025-07-13T09:01:52.407+05:30 INFO 23356 --- [spring-learn] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'

2025-07-13T09:01:53.738+05:30 INFO 23356 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 9090 (http)

2025-07-13T09:01:53.752+05:30 INFO 23356 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2025-07-13T09:01:53.752+05:30 INFO 23356 --- [spring-learn] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.42]

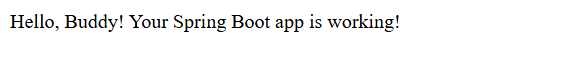
2025-07-13T09:01:53.786+05:30 INFO 23356 --- [spring-learn] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

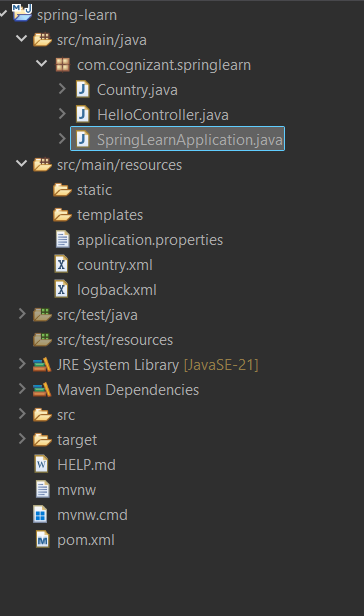
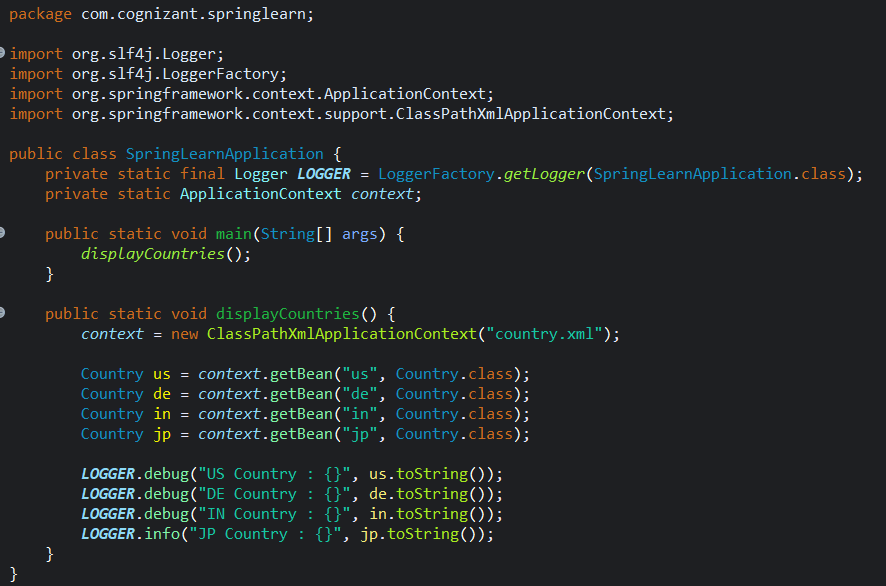
2025-07-13T09:01:53.787+05:30 INFO 23356 --- [spring-learn] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1378 ms

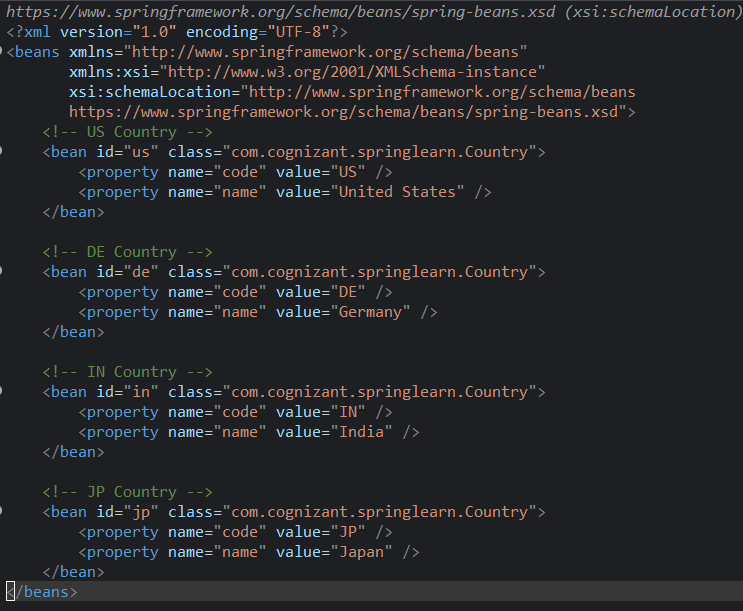
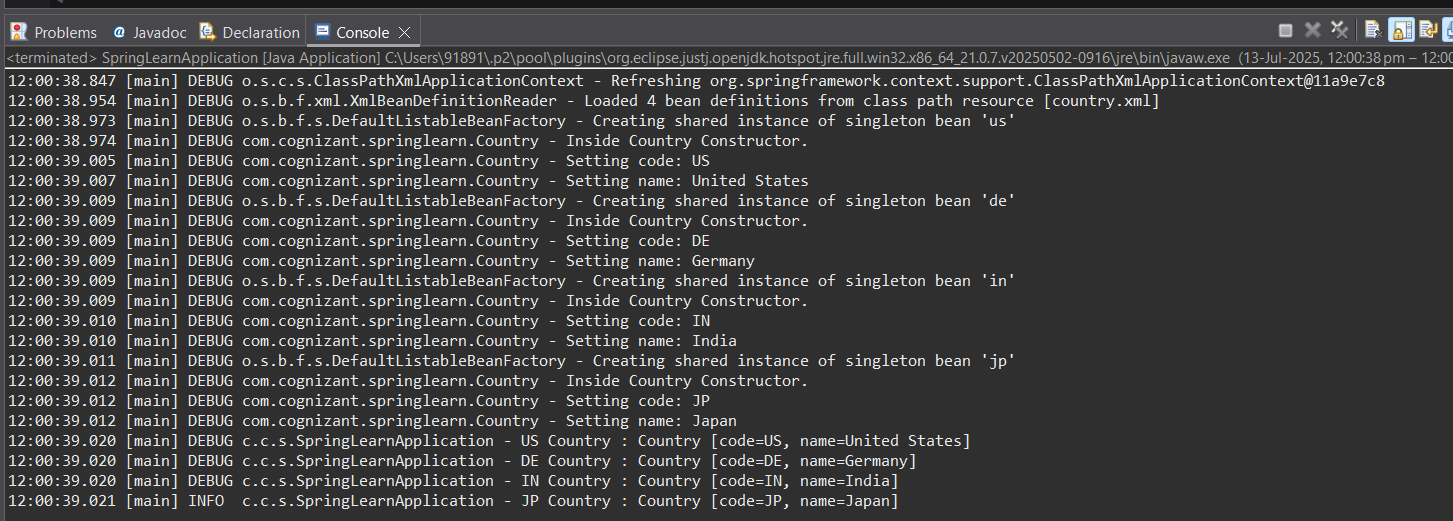
2025-07-13T09:01:54.266+05:30 INFO 23356 --- [spring-learn] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729

2025-07-13T09:01:54.331+05:30 INFO 23356 --- [spring-learn] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9090 (http) with context path '/'

2025-07-13T09:01:54.344+05:30 INFO 23356 --- [spring-learn] [ restartedMain] c.c.springlearn.SpringLearnApplication : Started SpringLearnApplication in 2.452 seconds (process running for 2.962)

The output of HelloController.java:

**2. Spring Core – Load Country from Spring Configuration XML**  
  
  
SpringLearnApplication.java:  


country.xml  
  
  
Country.java  
  
  
Output:  


**3. Hello World RESTful Web Service**

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);

}

}

HellosController.java

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 HellosController {

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

*@GetMapping*("/hellos")

public String sayHello() {

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

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

return "Hello World!!";

}

}

Output: Hello World!!

**4. REST – Country Web Service**

CountryController.java

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.Country;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

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

*@RestController*

public class CountryController {

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

*@RequestMapping*("/country")

public Country getCountryIndia() {

***LOGGER***.info("START - getCountryIndia()");

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

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

***LOGGER***.info("END - getCountryIndia()");

return country;

}

}

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);

}

}

Output:

{"code":"IN","name":"India"}

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

When the user sends a GET request to /country, the controller method getCountryIndia() is triggered. Inside this method, we load the country.xml file using Spring’s ClassPathXmlApplicationContext. We then retrieve the bean with ID "in" which contains the details for India (code and name), and return it directly.

Thanks to Spring Boot, this object is automatically converted to a JSON response and sent back to the client (browser or Postman).

**2. How is the bean converted into a JSON response?**

Spring Boot uses an internal library called **Jackson**, which is part of the spring-boot-starter-web dependency. When the controller method returns a Java object (like Country), Jackson steps in and serializes that object into JSON format.

We don’t have to write any code for this conversion—it happens automatically behind the scenes as long as we use annotations like @RestController or @ResponseBody.

**3. How to see HTTP header details in the browser (Network tab)?**

1. Open Chrome and go to http://localhost:8083/country.
2. Right-click and choose **Inspect**, then go to the **Network** tab.
3. Refresh the page. You’ll see the request to /country.
4. Click on it → go to the **Headers** tab.

You’ll see headers like:

* Content-Type: application/json
* Content-Length: 35
* Date: <current date>
* Server: Apache Tomcat/...

These headers help understand what type of data is sent and how the server handled the response.

**4. How to check HTTP headers in Postman?**

1. Open Postman and create a **GET** request to:  
   http://localhost:8083/country
2. Click **Send**.
3. Below the response, go to the **Headers** tab.

You’ll see similar headers as in the browser:

* Content-Type: application/json
* Content-Length
* Date
* Server

These headers confirm that our Spring Boot REST API is returning the expected data format and metadata.

**5. REST – Get Country based on country code**

CountryService.java

package com.cognizant.springlearn.service;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.stereotype.Service;

import com.cognizant.springlearn.Country;

*@Service*

public class CountryService {

private ApplicationContext context;

public Country getCountry(String code) {

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

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

return countryList.stream()

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

.findFirst()

.orElse(null); // You can throw an exception instead

}

}

CountryController.java

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.Country;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

import com.cognizant.springlearn.service.CountryService;

*@RestController*

public class CountryController {

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

*@Autowired*

private CountryService countryService;

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

public Country getCountry(*@PathVariable* String code) {

***LOGGER***.info("Start - getCountry");

Country country = countryService.getCountry(code);

***LOGGER***.info("End - getCountry");

return country;

}

}

Output:

<http://localhost:9090/countries/us> :

{

    "code": "US",

    "name": "United States"

}

<http://localhost:9090/countries/in> :

{

    "code": "IN",

    "name": "India"

}

<http://localhost:9090/countries/jp> :

{

    "code": "JP",

    "name": "Japan"

}

**6. Create authentication service that returns JWT**

JwtUtil.java

package com.cognizant.springlearn.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.security.Key;

*@Component*

public class JwtUtil {

private static final long ***EXPIRATION\_TIME*** = 1000 \* 60 \* 60 \* 10; // 10 hours

private final Key key = Keys.*secretKeyFor*(*SignatureAlgorithm*.***HS256***); // auto-generated key

public String generateToken(String username) {

return Jwts.*builder*()

.setSubject(username)

.setIssuedAt(new Date(System.*currentTimeMillis*()))

.setExpiration(new Date(System.*currentTimeMillis*() + ***EXPIRATION\_TIME***))

.signWith(key)

.compact();

}

}

SecurityConfig.java

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.web.SecurityFilterChain;

*@Configuration*

public class SecurityConfig {

*@Bean*

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.csrf(csrf -> csrf.disable())

.authorizeHttpRequests(auth -> auth

.requestMatchers("/authenticate").permitAll()

.anyRequest().authenticated()

);

return http.build();

}

}

AuthenticationController.java

package com.cognizant.springlearn.controller;

import jakarta.servlet.http.HttpServletRequest;

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

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

import com.cognizant.springlearn.util.JwtUtil;

import java.nio.charset.StandardCharsets;

import java.util.Base64;

*@RestController*

public class AuthenticationController {

*@Autowired*

private JwtUtil jwtUtil;

*@GetMapping*("/authenticate")

public String authenticate(HttpServletRequest request) {

String authHeader = request.getHeader("Authorization");

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

String base64Credentials = authHeader.substring("Basic ".length());

byte[] credDecoded = Base64.*getDecoder*().decode(base64Credentials);

String credentials = new String(credDecoded, StandardCharsets.***UTF\_8***);

final String[] values = credentials.split(":", 2);

String username = values[0];

String password = values[1];

if (username.equals("user") && password.equals("pwd")) {

String token = jwtUtil.generateToken(username);

return "{\"token\":\"" + token + "\"}";

}

}

throw new RuntimeException("Invalid Credentials");

}

}

Output:

<http://localhost:9090/authenticate>:

Username: user

Password: pwd

{

    "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNzUyMzk0NjcyLCJleHAiOjE3NTI0MzA2NzJ9.gUW0n4um4yWZ7d1dVFtopGOs3zqM\_pvNgOqZFIBe2zM"

}

