

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.

Process:

Step 1: Go to Spring Initializr

Open <https://start.spring.io/> in your browser.

Step 2: Configure Project Metadata

- ☐ Group: com.cognizant
- ☐ Artifact: spring-learn
- ☐ Name: spring-learn
- ☐ Packaging: Jar
- ☐ Java Version: Choose your version (preferably 17 or 21)
- ☐ Dependencies:

- ☐ Spring Boot DevTools
- ☐ Spring Web

Step 3: Generate the Project

- Click **Generate**.
- A .zip file will be downloaded (e.g., spring-learn.zip).

Step 4: Import Maven Project

1. Open Eclipse.
2. Go to: File > Import > Maven > Existing Maven Projects > Next.
3. **Browse** to the extracted spring-learn folder.
4. Click **Finish**.

Step 5. Add Logs in main() Method

```
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("SpringLearnApplication started successfully.");
    }
}
```

Step 6: Run the Application

- Right-click SpringLearnApplication.java > Run As > Java Application.
- Console should show SpringLearnApplication started successfully.

Out put:

```
SpringLearnApplication [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (7 Jul 2025, 11:02:13 am) [pid: 2816]
( ( ) \_ | ' | | | | ' \_ | \ \ \ \ )
 \ \_ | | | | | | | | | | | ) ) ) ) )
  | | | | | | | | | | | | | | | / / / / /
=====|_|=====|_|_/_/_/_/_/

:: Spring Boot ::                (v3.5.3)

2025-07-07T11:02:18.555+05:30 INFO 2816 --- [spring-learn] [ restartedMain] c.c.spring
2025-07-07T11:02:18.573+05:30 INFO 2816 --- [spring-learn] [ restartedMain] c.c.spring
2025-07-07T11:02:18.882+05:30 INFO 2816 --- [spring-learn] [ restartedMain] .e.DevTool
2025-07-07T11:02:18.882+05:30 INFO 2816 --- [spring-learn] [ restartedMain] .e.DevTool
2025-07-07T11:02:22.302+05:30 INFO 2816 --- [spring-learn] [ restartedMain] o.s.b.w.en
2025-07-07T11:02:22.365+05:30 INFO 2816 --- [spring-learn] [ restartedMain] o.apache.c
2025-07-07T11:02:22.366+05:30 INFO 2816 --- [spring-learn] [ restartedMain] o.apache.c
2025-07-07T11:02:22.561+05:30 INFO 2816 --- [spring-learn] [ restartedMain] o.a.c.c.C.
2025-07-07T11:02:22.565+05:30 INFO 2816 --- [spring-learn] [ restartedMain] w.s.c.Serv
2025-07-07T11:02:24.148+05:30 INFO 2816 --- [spring-learn] [ restartedMain] o.s.b.d.a.
2025-07-07T11:02:24.311+05:30 INFO 2816 --- [spring-learn] [ restartedMain] o.s.b.w.en
2025-07-07T11:02:24.363+05:30 INFO 2816 --- [spring-learn] [ restartedMain] c.c.spring
SpringLearnApplication started successfully!
```

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

Process:

Step 1. Create Country.java

```
package com.cognizant.springlearn;

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;

    public Country() {

        LOGGER.debug("Inside Country Constructor.");

    }

    public String getCode() {

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

        return code;

    }

    public void setCode(String code) {

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

        this.code = code;

    }

    public String getName() {

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

        return name;

    }

    public void setName(String name) {

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

        this.name = name;

    }

}
```

@Override

```
public String toString() {  
    return "Country [code=" + code + ", name=" + name + "];"  
}  
}
```

Step 2. Create country.xml Configuration 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">  
  
    <bean id="country" class="com.cognizant.springlearn.Country">  
        <property name="code" value="IN"/>  
        <property name="name" value="India"/>  
    </bean>  
  
</beans>
```

Step 3. Modify SpringLearnApplication.java

```
package com.cognizant.springlearn;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class SpringLearnApplication {  
  
    private static final Logger LOGGER =  
        LoggerFactory.getLogger(SpringLearnApplication.class);  
  
    public static void displayCountry() {  
        ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
        Country country = context.getBean("country", Country.class);  
        LOGGER.debug("Country : {}", country.toString());  
    }  
  
    public static void main(String[] args) {  
        LOGGER.debug("START");  
    }  
}
```

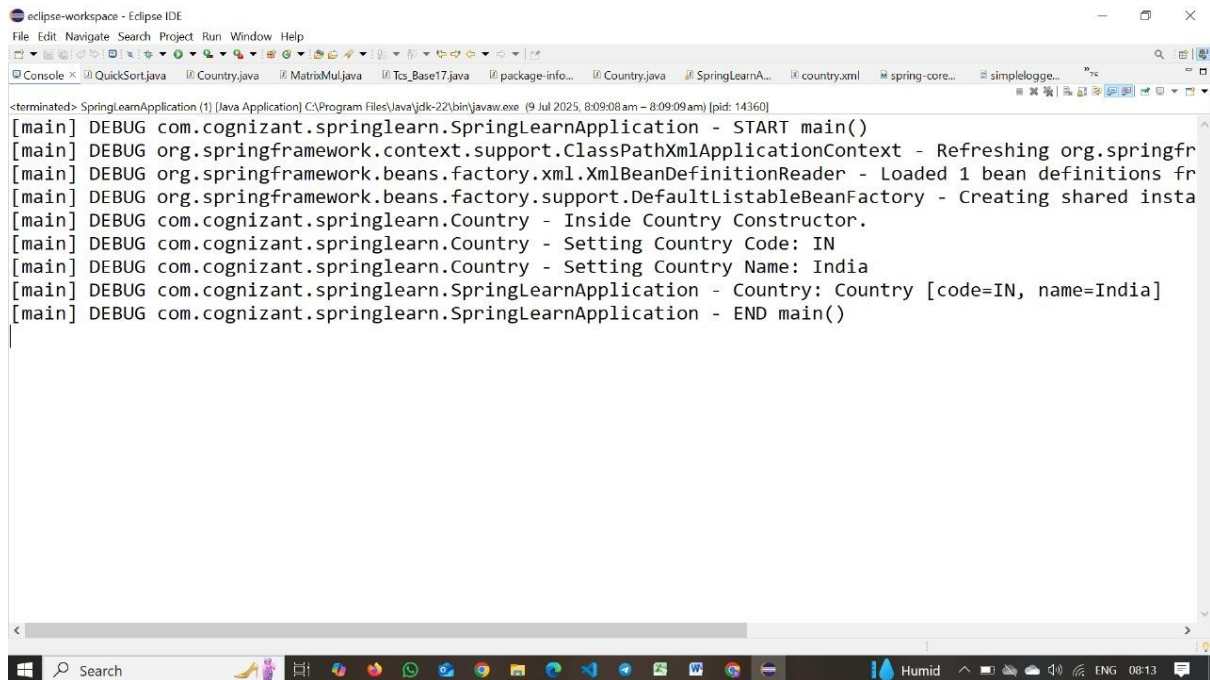
```

        displayCountry();

        LOGGER.debug("END");
    }
}

```

Step 4: Out Put



```

<terminated> SpringLearnApplication (1) [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (9 Jul 2025, 8:09:08 am - 8:09:09 am) [pid: 14360]
[main] DEBUG com.cognizant.springlearn.SpringLearnApplication - START main()
[main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springfr
[main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 1 bean definitions fr
[main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared insta
[main] DEBUG com.cognizant.springlearn.Country - Inside Country Constructor.
[main] DEBUG com.cognizant.springlearn.Country - Setting Country Code: IN
[main] DEBUG com.cognizant.springlearn.Country - Setting Country Name: India
[main] DEBUG com.cognizant.springlearn.SpringLearnApplication - Country: Country [code=IN, name=India]
[main] DEBUG com.cognizant.springlearn.SpringLearnApplication - END main()

```

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.

Process:

Step 1: Create HelloController.java

```
package com.cognizant.spring_learn.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.debug("START: sayHello()");

        String message = "Hello World!!";

        LOGGER.debug("END: sayHello()");

        return message;

    }

}
```

Step 2: Start Your Spring Boot Application

```
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);

    }

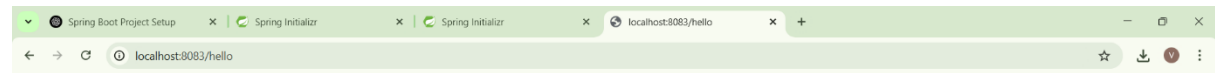
}
```



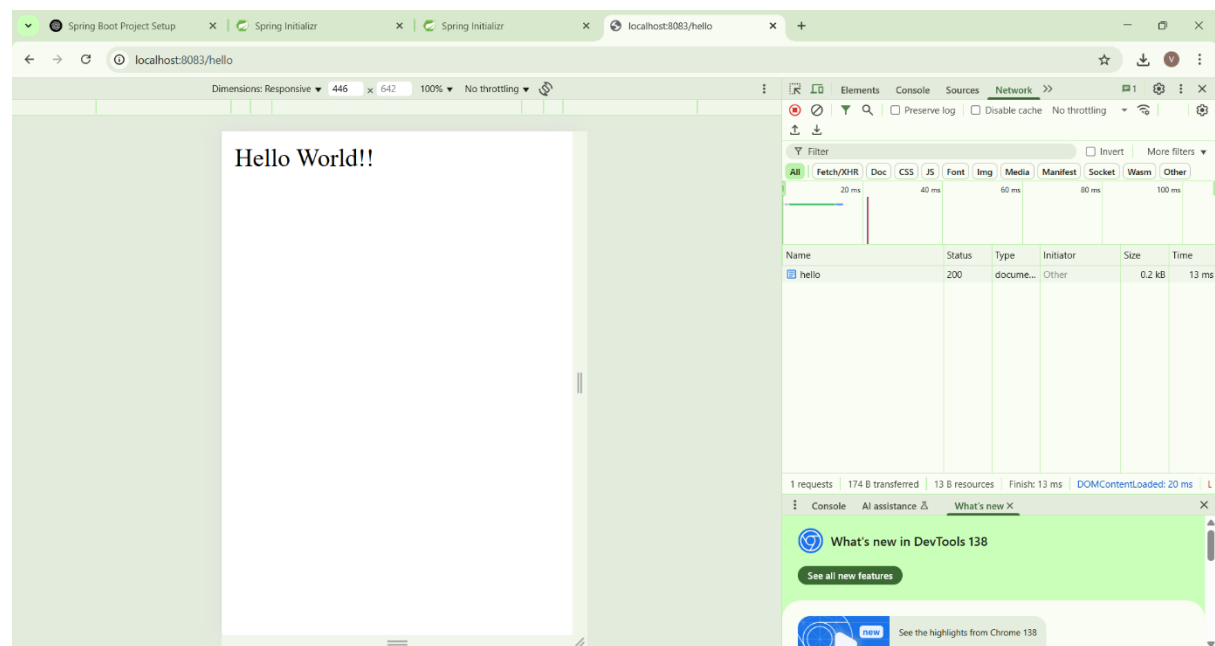
```
}  
  
}
```

Step 3: Use a Browser

<http://localhost:8080/hello>



Hello World!!



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 response?
- 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

Process:

Step 1: Add CountryController.java

```
package com.cognizant.spring_learn.controller;

import com.cognizant.spring_learn.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.debug("START: getCountryIndia");
    ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
    Country country = context.getBean("country", Country.class);
    LOGGER.debug("Country returned: {}", country);
    return country; // Spring will convert it to JSON
}
}

```

Step 2: country.xml Is Created

```

<?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">
  <bean id="country" class="com.cognizant.spring_learn.Country">
    <property name="code" value="IN"/>
    <property name="name" value="India"/>
  </bean>
</beans>

```

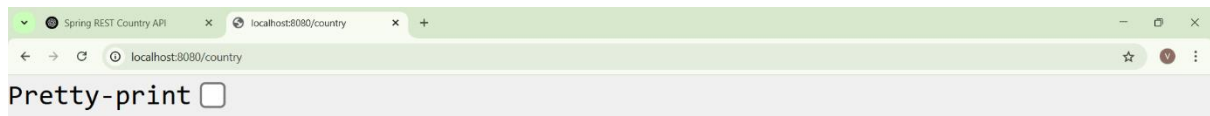
Step 3: Run the Spring Boot Application

```

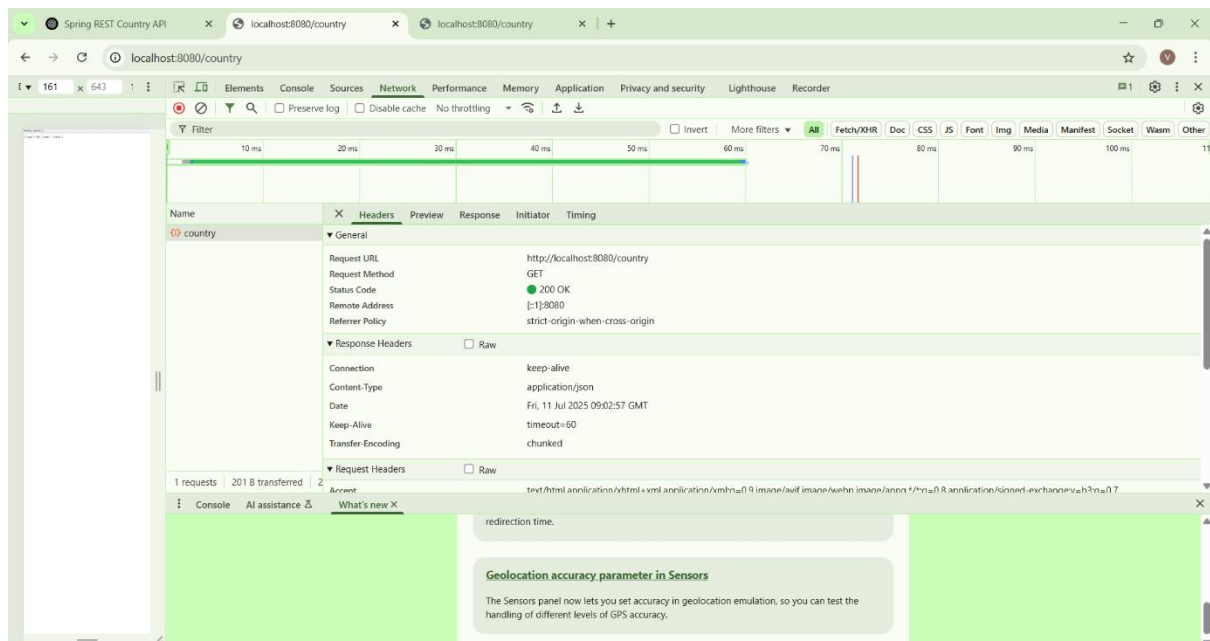
@SpringBootApplication
public class SpringLearnApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringLearnApplication.class, args);
    }
}

```

Step 4: Using Browser and View Headers in Browser Dev Tools (F12)



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



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 Implementation: 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"
}
```

Process:

Step 1: country.xml (in src/main/resources)

```
<?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>
        <bean class="com.cognizant.spring_learn.model.Country">
          <property name="code" value="IN"/>
          <property name="name" value="India"/>
        </bean>
        <bean class="com.cognizant.spring_learn.model.Country">
          <property name="code" value="US"/>
          <property name="name" value="United States"/>
        </bean>
        <bean class="com.cognizant.spring_learn.model.Country">
          <property name="code" value="DE"/>
```

```

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

    <bean class="com.cognizant.spring_learn.model.Country">
        <property name="code" value="JP"/>
        <property name="name" value="Japan"/>
    </bean>

</list>

</constructor-arg>

</bean>

</beans>

```

Step 2: Country.java (com.cognizant.spring_learn.model)

```

package com.cognizant.spring_learn.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;

    public Country() {
        LOGGER.debug("Inside Country Constructor.");
    }

    public String getCode() {
        LOGGER.debug("Getting code");
        return code;
    }

    public void setCode(String code) {
        LOGGER.debug("Setting code: {}", code);
        this.code = code;
    }

    public String getName() {
        LOGGER.debug("Getting name");
        return name;
    }
}

```

```

    }

    public void setName(String name) {
        LOGGER.debug("Setting name: {}", name);
        this.name = name;
    }

    @Override
    public String toString() {
        return "Country [code=" + code + ", name=" + name + "]";
    }
}

```

Step 3: CountryService.java (com.cognizant.spring_learn.service)

```

package com.cognizant.spring_learn.service;

import com.cognizant.spring_learn.model.Country;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class CountryService {

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

    public Country getCountry(String code) {
        LOGGER.debug("Start getCountry with code: {}", code);
        ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
        List<Country> countryList = (List<Country>) context.getBean("countryList");
        // Lambda expression for case-insensitive match
        return countryList.stream()
            .filter(c -> c.getCode().equalsIgnoreCase(code))
            .findFirst()
            .orElse(null); // or throw custom CountryNotFoundException
    }
}

```

```
}
```

Step 4: CountryController.java (com.cognizant.spring_learn.controller)

```
package com.cognizant.spring_learn.controller;

import com.cognizant.spring_learn.model.Country;
import com.cognizant.spring_learn.service.CountryService;
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 {

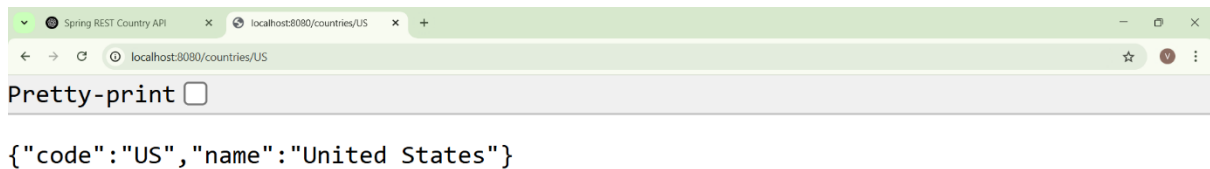
    private static final Logger LOGGER = LoggerFactory.getLogger(CountryController.class);

    @Autowired
    private CountryService countryService;

    @GetMapping("/country/{code}")
    public Country getCountry(@PathVariable String code) {
        LOGGER.debug("Start getCountry() in controller with code: {}", code);
        Country country = countryService.getCountry(code);
        LOGGER.debug("End getCountry() in controller");
        return country;
    }
}
```

Step 5: Run Your Application

<http://localhost:8083/country/US>



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.eyJzdWUiOiJ1c2VyliwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOiE1NzAzODA2NzR9.t3LRvICV-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.

Process:

Step 1: Project Setup and Dependencies

```
<!-- Spring Boot Starter Web -->
```

```
<dependency>
```

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

```
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

```
<!-- JWT dependencies -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
```

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

Step 2: Create SecurityConfig to accept Basic Auth on /authenticate

```
package com.cognizant.spring_learn.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
```

```

import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/authenticate").permitAll()
                .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults()); // Enables Basic Auth parsing
        return http.build();
    }
}

```

Step 3: Create AuthenticationController

```

package com.cognizant.spring_learn.controller;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

```

```

@RestController

public class AuthenticationController {

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

    private static final Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256); // Secret key
    for signing

    @GetMapping("/authenticate")

        public ResponseEntity<Map<String, String>> authenticate() {

            LOGGER.info("Inside /authenticate");

            // Get authenticated user's username from SecurityContext

            Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

            String username = authentication.getName();

            // For demo: just accept if username == "user" and password is "pwd"

            if (!"user".equals(username)) {

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

            }

            // Generate JWT token valid for 15 minutes

            long nowMillis = System.currentTimeMillis();

            Date now = new Date(nowMillis);

            Date exp = new Date(nowMillis + 15 * 60 * 1000);

            String jwt = Jwts.builder()

                .setSubject(username)

                .setIssuedAt(now)

                .setExpiration(exp)

                .signWith(key)

                .compact();

            LOGGER.info("Generated Token: {}", jwt);

            Map<String, String> tokenMap = new HashMap<>();

            tokenMap.put("token", jwt);

            return ResponseEntity.ok(tokenMap);

        }

    }
}

```

Step 4: Out put

