**Superset ID: 6394725**

**Spring REST using Spring Boot 3**

**Hands on 1**

**Create a Spring Web Project using Maven**

**Creating the Spring Web Project:**

1. **Go to** [**https://start.spring.io/**](https://start.spring.io/)

* This is Spring Initializr – a project generator.

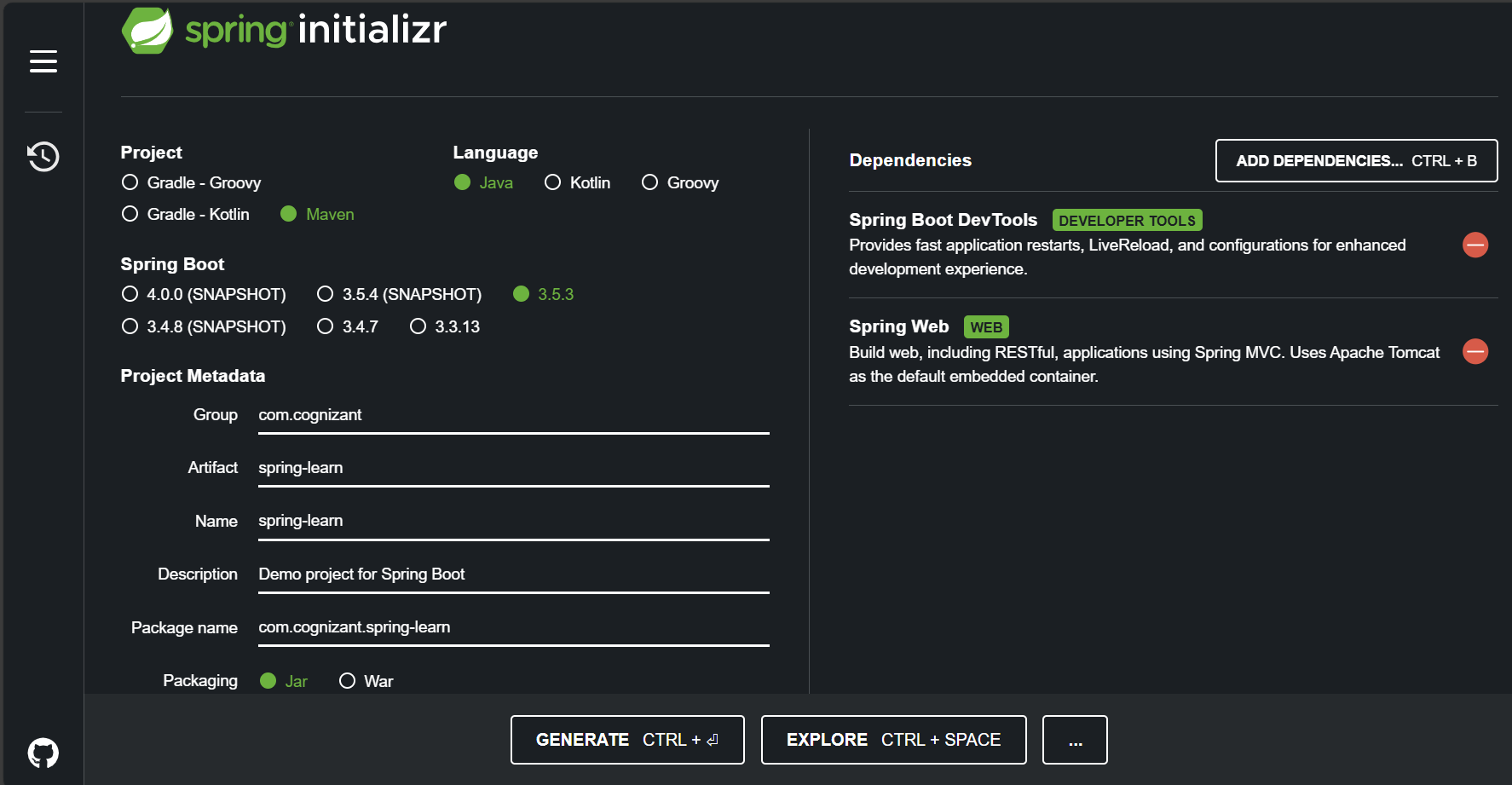
1. **Change:**

* **Group** → com.cognizant
* **Artifact** → spring-learn

1. **Add Dependencies:**

* Spring Boot DevTools
* Spring Web

1. Click **Generate**, and download the .zip file.



### **Setup in Eclipse:**

1. Extract the .zip file to your Eclipse Workspace directory.
2. Open Eclipse →File > Import > Maven > Existing Maven Projects > Browse to extracted folder > Finish.

**Add Logs and Run Application**

1. Open SpringLearnApplication.java (in src/main/java/com/cognizant/springlearn)
2. Add a log message inside the main() method:

package com.cognizant.spring\_learn;

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) {

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

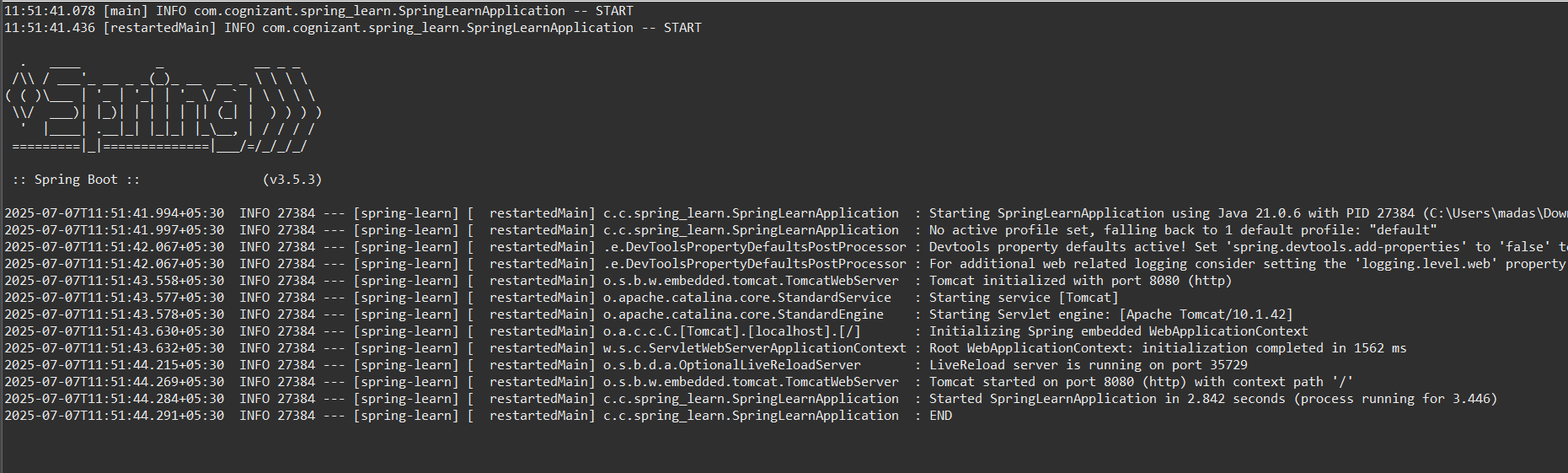
SpringApplication.*run*(SpringLearnApplication.class, args);

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

}

}

1. Run SpringLearnApplication as a **Java Application**.

**SME Walkthrough Points**

#### **1. src/main/java**

* Contains application source code including the main class SpringLearnApplication.java.
* All components like Controllers, Services, and Repositories go here.

#### **2. src/main/resources**

* Contains configuration files like:
  + application.properties
  + Static files like HTML, CSS if required.
  + logback.xml (if logging config is needed)

#### **3. src/test/java**

* Used for writing unit tests using JUnit or Mockito.

#### **4. SpringLearnApplication.java**

* Entry point of Spring Boot Application.
* Contains main() method.
* Calls SpringApplication.run() to bootstrap the application.

#### **5. @SpringBootApplication Annotation**

* Combines three annotations:
  + @Configuration – Marks class as a source of bean definitions.
  + @EnableAutoConfiguration – Enables Spring Boot’s auto-configuration.
  + @ComponentScan – Scans for components in the package and sub-packages.

**6. pom.xml**

* Defines project dependencies and plugins.

**Dependency Hierarchy View:**

Downloaded from central: https://repo.maven.apache.org/maven2/com/github/luben/zstd-jni/1.5.6-3/zstd-jni-1.5.6-3.jar (6.7 MB at 42 kB/s)

[INFO] com.cognizant:spring-learn:jar:0.0.1-SNAPSHOT

[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:3.5.3:compile

[INFO] | +- org.springframework.boot:spring-boot-starter:jar:3.5.3:compile

[INFO] | | +- org.springframework.boot:spring-boot-starter-logging:jar:3.5.3:compile

[INFO] | | | +- ch.qos.logback:logback-classic:jar:1.5.18:compile

[INFO] | | | | \- ch.qos.logback:logback-core:jar:1.5.18:compile

[INFO] | | | +- org.apache.logging.log4j:log4j-to-slf4j:jar:2.24.3:compile

[INFO] | | | | \- org.apache.logging.log4j:log4j-api:jar:2.24.3:compile

[INFO] | | | \- org.slf4j:jul-to-slf4j:jar:2.0.17:compile

[INFO] | | +- jakarta.annotation:jakarta.annotation-api:jar:2.1.1:compile

[INFO] | | \- org.yaml:snakeyaml:jar:2.4:compile

[INFO] | +- org.springframework.boot:spring-boot-starter-json:jar:3.5.3:compile

[INFO] | | +- com.fasterxml.jackson.core:jackson-databind:jar:2.19.1:compile

[INFO] | | | +- com.fasterxml.jackson.core:jackson-annotations:jar:2.19.1:compile

[INFO] | | | \- com.fasterxml.jackson.core:jackson-core:jar:2.19.1:compile

[INFO] | | +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.19.1:compile

[INFO] | | +- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.19.1:compile

[INFO] | | \- com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.19.1:compile

[INFO] | +- org.springframework.boot:spring-boot-starter-tomcat:jar:3.5.3:compile

[INFO] | | +- org.apache.tomcat.embed:tomcat-embed-core:jar:10.1.42:compile

[INFO] | | +- org.apache.tomcat.embed:tomcat-embed-el:jar:10.1.42:compile

[INFO] | | \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:10.1.42:compile

[INFO] | +- org.springframework:spring-web:jar:6.2.8:compile

[INFO] | | +- org.springframework:spring-beans:jar:6.2.8:compile

[INFO] | | \- io.micrometer:micrometer-observation:jar:1.15.1:compile

[INFO] | | \- io.micrometer:micrometer-commons:jar:1.15.1:compile

[INFO] | \- org.springframework:spring-webmvc:jar:6.2.8:compile

[INFO] | +- org.springframework:spring-aop:jar:6.2.8:compile

[INFO] | +- org.springframework:spring-context:jar:6.2.8:compile

[INFO] | \- org.springframework:spring-expression:jar:6.2.8:compile

[INFO] +- org.springframework.boot:spring-boot-devtools:jar:3.5.3:runtime (optional)

[INFO] | +- org.springframework.boot:spring-boot:jar:3.5.3:compile

[INFO] | \- org.springframework.boot:spring-boot-autoconfigure:jar:3.5.3:compile

[INFO] \- org.springframework.boot:spring-boot-starter-test:jar:3.5.3:test

[INFO] +- org.springframework.boot:spring-boot-test:jar:3.5.3:test

[INFO] +- org.springframework.boot:spring-boot-test-autoconfigure:jar:3.5.3:test

[INFO] +- com.jayway.jsonpath:json-path:jar:2.9.0:test

[INFO] | \- org.slf4j:slf4j-api:jar:2.0.17:compile

[INFO] +- jakarta.xml.bind:jakarta.xml.bind-api:jar:4.0.2:test

[INFO] | \- jakarta.activation:jakarta.activation-api:jar:2.1.3:test

[INFO] +- net.minidev:json-smart:jar:2.5.2:test

[INFO] | \- net.minidev:accessors-smart:jar:2.5.2:test

[INFO] | \- org.ow2.asm:asm:jar:9.7.1:test

[INFO] +- org.assertj:assertj-core:jar:3.27.3:test

[INFO] | \- net.bytebuddy:byte-buddy:jar:1.17.6:test

[INFO] +- org.awaitility:awaitility:jar:4.2.2:test

[INFO] +- org.hamcrest:hamcrest:jar:3.0:test

[INFO] +- org.junit.jupiter:junit-jupiter:jar:5.12.2:test

[INFO] | +- org.junit.jupiter:junit-jupiter-api:jar:5.12.2:test

[INFO] | | +- org.opentest4j:opentest4j:jar:1.3.0:test

[INFO] | | +- org.junit.platform:junit-platform-commons:jar:1.12.2:test

[INFO] | | \- org.apiguardian:apiguardian-api:jar:1.1.2:test

[INFO] | +- org.junit.jupiter:junit-jupiter-params:jar:5.12.2:test

[INFO] | \- org.junit.jupiter:junit-jupiter-engine:jar:5.12.2:test

[INFO] | \- org.junit.platform:junit-platform-engine:jar:1.12.2:test

[INFO] +- org.mockito:mockito-core:jar:5.17.0:test

[INFO] | +- net.bytebuddy:byte-buddy-agent:jar:1.17.6:test

[INFO] | \- org.objenesis:objenesis:jar:3.3:test

[INFO] +- org.mockito:mockito-junit-jupiter:jar:5.17.0:test

[INFO] +- org.skyscreamer:jsonassert:jar:1.5.3:test

[INFO] | \- com.vaadin.external.google:android-json:jar:0.0.20131108.vaadin1:test

[INFO] +- org.springframework:spring-core:jar:6.2.8:compile

[INFO] | \- org.springframework:spring-jcl:jar:6.2.8:compile

[INFO] +- org.springframework:spring-test:jar:6.2.8:test

[INFO] \- org.xmlunit:xmlunit-core:jar:2.10.2:test

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 08:21 min

[INFO] Finished at: 2025-07-07T12:13:51+05:30

[INFO] ------------------------------------------------------------------------

**Hands on 2**

**Spring Core – Load SimpleDateFormat from Spring Configuration XML**

## **Create date-format.xml in src/main/resources**

1. Right-click on src/main/resources → New > File → name it: date-format.xml
2. In date-format.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*](http://www.springframework.org/schema/beans/spring-beans.xsd)*"*>

<**bean** id=*"dateFormat"* class=*"java.text.SimpleDateFormat"*>

<**constructor-arg** value=*"dd/MM/yyyy"* />

</**bean**>

</**beans**>

**Add displayDate() Method in SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.text.SimpleDateFormat;

import java.util.Date;

*@SpringBootApplication*

public class SpringLearnApplication {

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

public static void main(String[] args) throws Exception {

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

SpringApplication.*run*(SpringLearnApplication.class, args);

*displayDate*(); // <-- Call the date bean test method

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

}

public static void displayDate() throws Exception {

ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");

SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);

Date parsedDate = format.parse("31/12/2018");

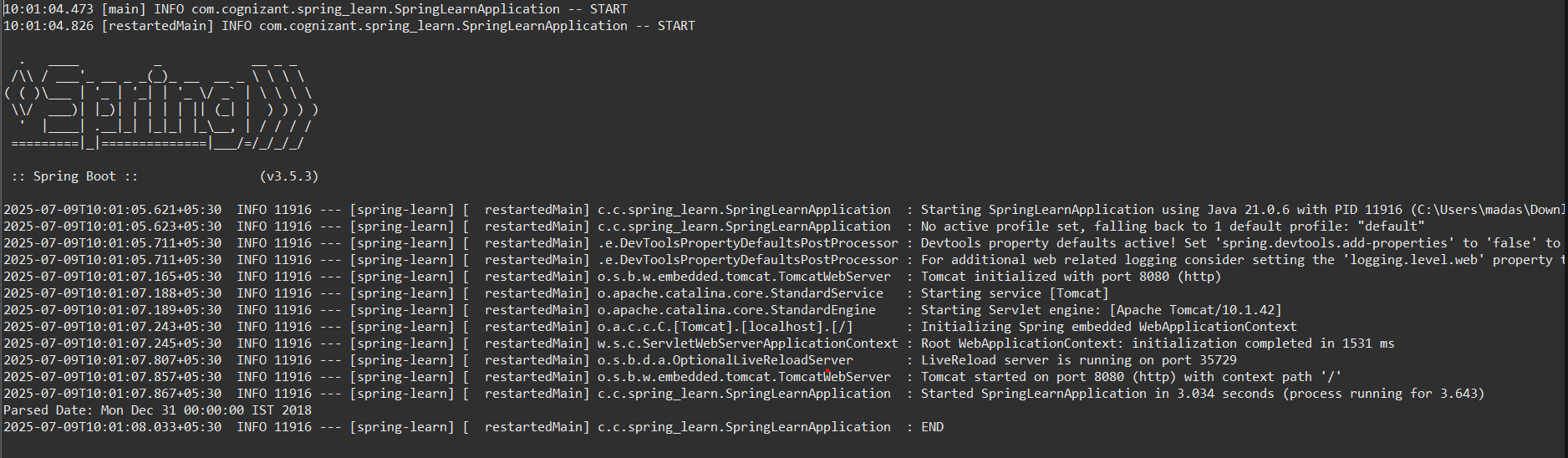
System.***out***.println("Parsed Date: " + parsedDate);

}

}

## **Run the Application**

1. Right-click SpringLearnApplication.java → Run As → Java Application.
2. Output in the console:



**Hands on 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

### **Step-by-Step Implementation:**

### **Create a new controller class**

Create this file: src/main/java/com/cognizant/spring\_learn/controller/HelloController.java

***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***.info("START - /hello called");

String response = "Hello World!!";

***LOGGER***.info("END - /hello called");

return response;

}

}

### **Set the server port to 8083**

Edit your application.properties file under: src/main/resources/application.properties

server.port=8083

spring.application.name=spring-learn

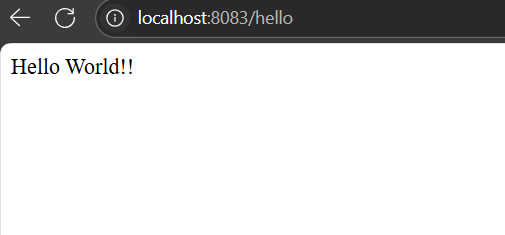
server.port=8083

### **Run the application**

### Right-click on SpringLearnApplication.java → Run As > Java Application.

**Test the endpoint**

**In Browser:** Go to: <http://localhost:8083/hello>Output:



**Hands on 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

**Step-by-Step Implementation**

### **Define the Country Class (POJO)**

### Create this file: src/main/java/com/cognizant/spring\_learn/model/Country.java

***Country.java***

package com.cognizant.spring\_learn.model;

public class Country {

private String code;

private String name;

// No-arg constructor (required for JSON serialization)

public Country() {}

public Country(String code, String name) {

this.code = code;

this.name = name;

}

// Getters and Setters

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;

}

}

### **Update date-format.xml to include a Country Bean**

src/main/resources/date-format.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*](http://www.springframework.org/schema/beans/spring-beans.xsd)*"*>

<!-- SimpleDateFormat Bean -->

<**bean** id=*"dateFormat"* class=*"java.text.SimpleDateFormat"*>

<**constructor-arg** value=*"dd/MM/yyyy"* />

</**bean**>

<!-- Country Bean -->

<**bean** id=*"country"* class=*"com.cognizant.spring\_learn.model.Country"*>

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

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

</**bean**>

</**beans**>

### **Create CountryController** src/main/java/com/cognizant/spring\_learn/controller/CountryController.java

package com.cognizant.spring\_learn.controller;

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.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 - /country called");

ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml"); // or country.xml

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

***LOGGER***.info("END - /country called");

return country;

}

}

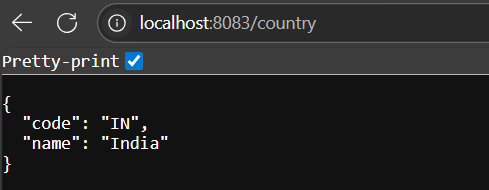
### **Run the App and Test**

Ensure this in application.properties:

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

Then:

* Run SpringLearnApplication.java
* Go to browser or Postman and visit: <http://localhost:8083/country>



**Hands on 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"

}

## **Step-by-Step Implementation**

### **Update country.xml with Country List**

src/main/resources/date-format.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*](http://www.springframework.org/schema/beans/spring-beans.xsd)*"*>

<**bean** id=*"country"* class=*"com.cognizant.spring\_learn.model.Country"*>

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

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

</**bean**>

<**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**>

</**list**>

</**constructor-arg**>

</**bean**>

</**beans**>

### **Create CountryService Class**

src/main/java/com/cognizant/spring\_learn/service/CountryService.java

package com.cognizant.spring\_learn.service;

import com.cognizant.spring\_learn.model.Country;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.stereotype.Service;

import java.util.List;

*@Service*

public class CountryService {

public Country getCountry(String code) {

ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");

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

return countryList.stream()

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

.findFirst()

.orElse(null); // You can throw an exception if not found

}

}

### **Update CountryController**

src/main/java/com/cognizant/spring\_learn/controller/CountryController.java

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*("/countries/{code}")

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

***LOGGER***.info("START - /countries/{}", code);

Country country = countryService.getCountry(code);

***LOGGER***.info("END - /countries/{}", code);

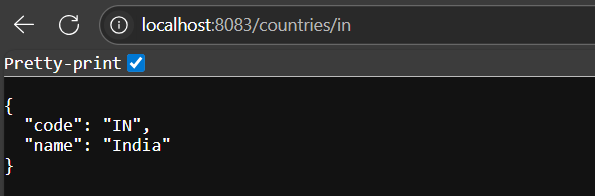
return country;

}

}

## **Test in Browser or Postman**

### **URL:** <http://localhost:8083/countries/in>



**Hands on 5**

**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.

**Step-by-step Implementation**

## **Overview: Authentication Service That Returns JWT**

When we execute this: curl -s -u user:pwd <http://localhost:8090/authenticate>

It should:

1. Read the credentials from the **Basic Auth header**.
2. Validate the credentials.
3. Generate and return a **JWT token**.

**Step-1: Update pom.xml with correct JWT dependencies:**

<dependencies>

<!-- Spring Boot Web Starter -->

<dependency>

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

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

</dependency>

<!-- Spring Boot DevTools -->

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- Spring Boot Testing -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<!-- JSON Web Token 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>

</dependencies>

**Step-2: Create JwtUtil.java**

package com.cognizant.spring\_learn.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import java.security.Key;

import java.util.Date;

public class JwtUtil {

private static final Key ***SECRET\_KEY*** = Keys.*hmacShaKeyFor*("mysecretkeymysecretkeymysecretkey".getBytes());

public static String generateToken(String username) {

long nowMillis = System.*currentTimeMillis*();

long expMillis = nowMillis + 1000 \* 60 \* 60;

return Jwts.*builder*()

.setSubject(username)

.setIssuedAt(new Date(nowMillis))

.setExpiration(new Date(expMillis))

.signWith(***SECRET\_KEY***, *SignatureAlgorithm*.***HS256***)

.compact();

}

}

**Step-3: Create Authentication Controller**

src/main/java/com/cognizant/spring\_learn /controller/AuthenticationController.java

package com.cognizant.spring\_learn.controller;

import com.cognizant.spring\_learn.util.JwtUtil;

import org.springframework.http.ResponseEntity;

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

import jakarta.servlet.http.HttpServletRequest;

import java.util.Base64;

*@RestController*

public class AuthenticationController {

private static final String ***VALID\_USERNAME*** = "user";

private static final String ***VALID\_PASSWORD*** = "pwd";

*@GetMapping*("/authenticate")

public ResponseEntity<?> authenticate(HttpServletRequest request) {

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

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

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

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

String credentials = new String(decodedBytes);

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

String username = values[0];

String password = values[1];

if (***VALID\_USERNAME***.equals(username) && ***VALID\_PASSWORD***.equals(password)) {

String token = JwtUtil.*generateToken*(username);

return ResponseEntity.*ok*().body("{\"token\":\"" + token + "\"}");

}

}

return ResponseEntity.*status*(401).body("Invalid credentials");

}

}

**Step-4: Set application.properties port**

server.port=8090

**Step-5: Run Application and Test with curl**

Open your terminal (not Eclipse terminal), and run:

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

**Output:**

