**Spring REST using Spring Boot 3**

**Create a Spring Web Project using Maven**

**Step 1: Create Spring Boot Project**

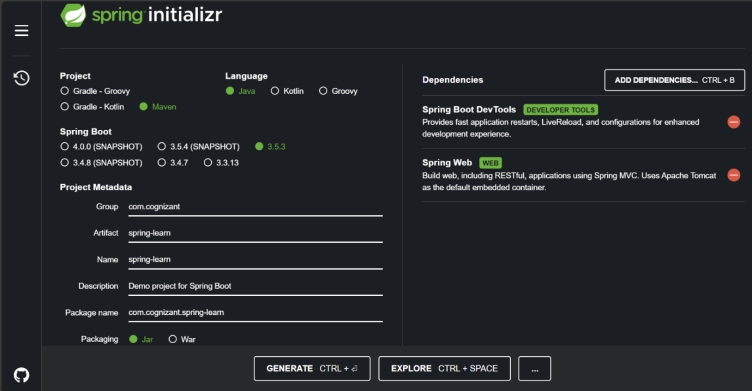
1. Open a browser and go to: <https://start.spring.io>
2. Set the following values:

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

1. Select dependencies:

* **Spring Boot DevTools**
* **Spring Web**

1. Click on **Generate** to download the project as a ZIP file.



Step 2: Extract and Import Project

1. Extract the ZIP file to your Eclipse workspace folder.
2. Open Eclipse IDE.
3. Go to **File > Import > Maven > Existing Maven Projects**
4. Click **Browse**, select the extracted folder, and click **Finish**.

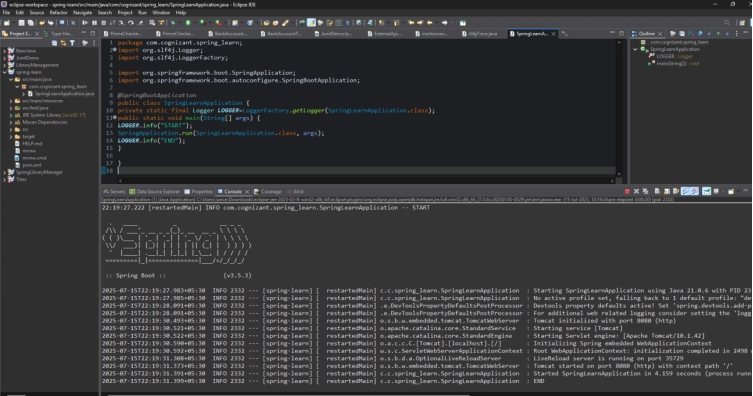
**Step 3: Build the Project**

1. Open Command Prompt or Terminal.
2. Navigate to the root folder of the project.
3. Run the following Maven command to build the project:

mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456

**Step 4: Add Logs and Run Application**

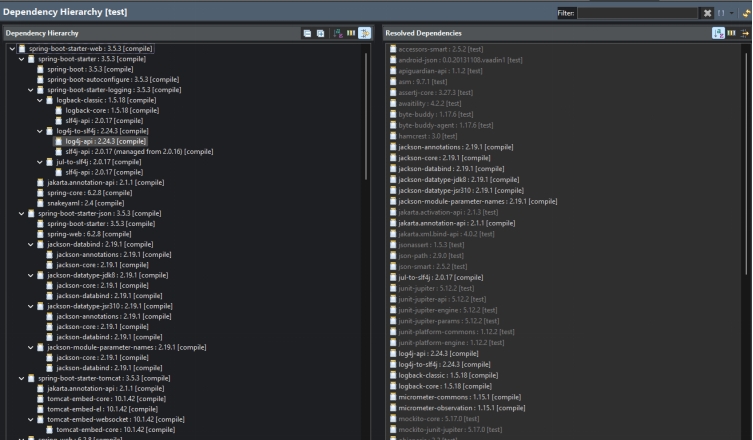
1. Open SpringLearnApplication.java in Eclipse.
2. Add a log message inside the main() method:
3. Run SpringLearnApplication as a **Java Application**.
4. Verify logs in the console. You should see:



**Code and Project Structure Walkthrough**

1. **src/main/java**
   * Contains application source code.
   * Example: com.cognizant.springlearn.SpringLearnApplication.java
2. **src/main/resources**
   * Stores configuration files like application.properties.
3. **src/test/java**
   * Contains test cases for your application.
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:
     1. @Configuration – Marks class as a source of bean definitions.
     2. @EnableAutoConfiguration – Enables Spring Boot’s auto-configuration.
     3. @ComponentScan – Scans for components in the package and sub-packages.
6. **pom.xml**
   * Defines project dependencies and plugins.

**Dependency Hierarchy View:**



**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:

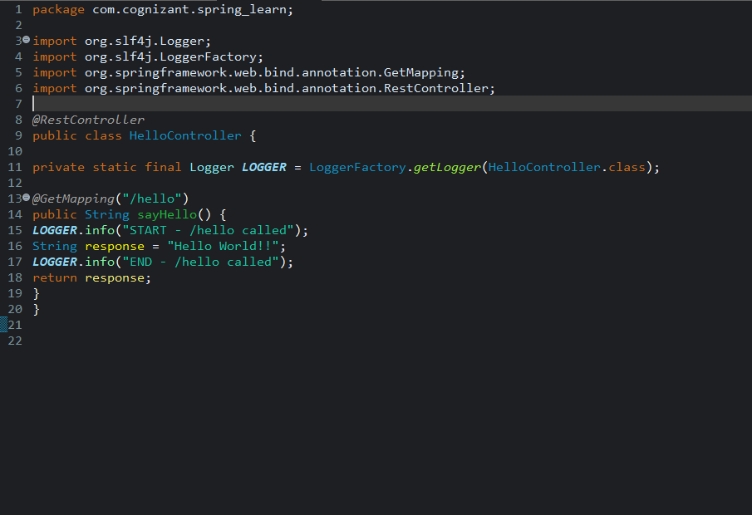
* In network tab of developer tools show the HTTP header details received
* In postman click on "Headers" tab to view the HTTP header details received

**Step-by-Step Implementation:**

**Create a new controller class**

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

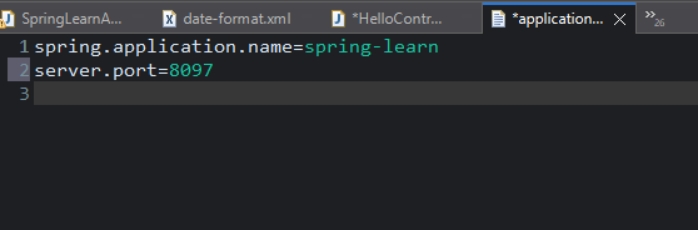
***HelloController.java***



**Set the server port to 8083**

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

server.port=8097

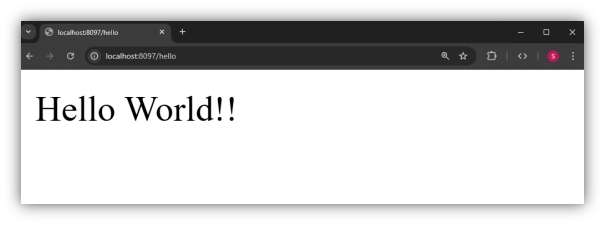


**Run the application**

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

**Test the endpoint**

**In Browser:** Go to: <http://localhost:8097/hello>Output:



**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:

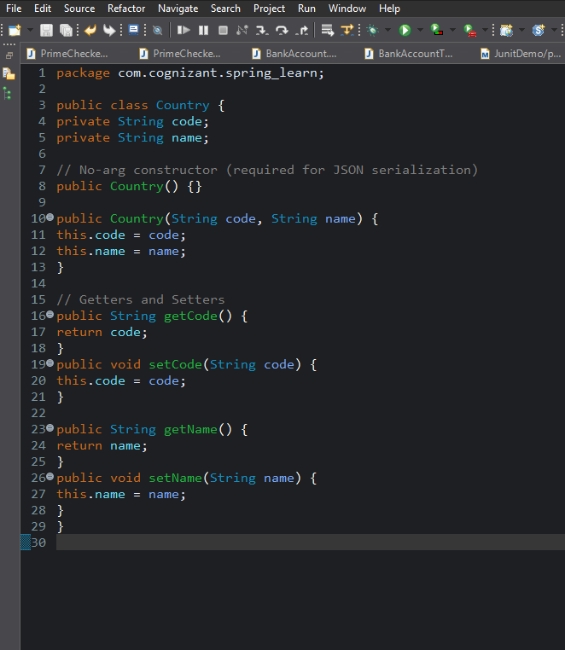
* What happens in the controller method?
* How the bean is converted into JSON reponse?
* In network tab of developer tools show the HTTP header details received
* In postman click on "Headers" tab to view the HTTP header details received

**Step-by-Step Implementation**

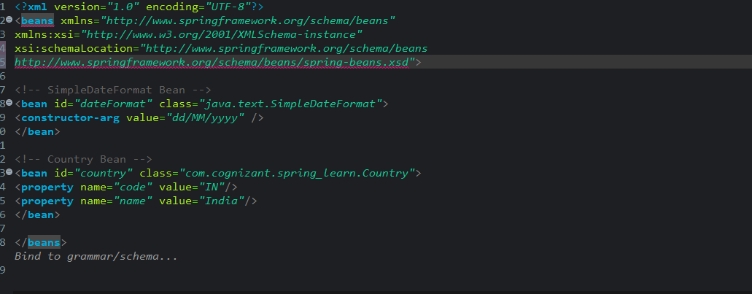
**Define the Country Class**

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

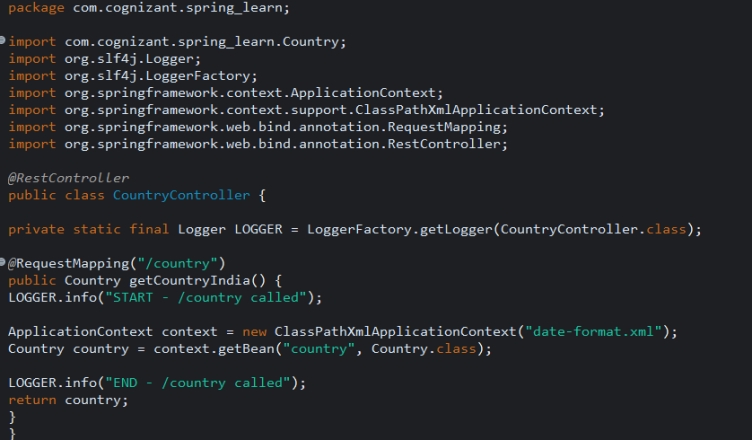
***Country.java***



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



**Create CountryController**

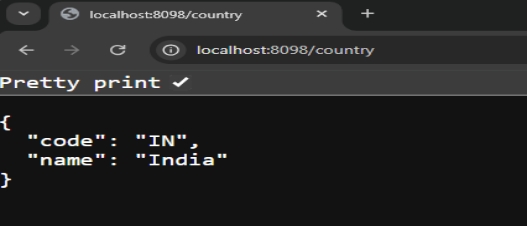


**Run the App and Test**

server.port=8098  
spring.application.name=spring-learn

Then:

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



**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**:

{

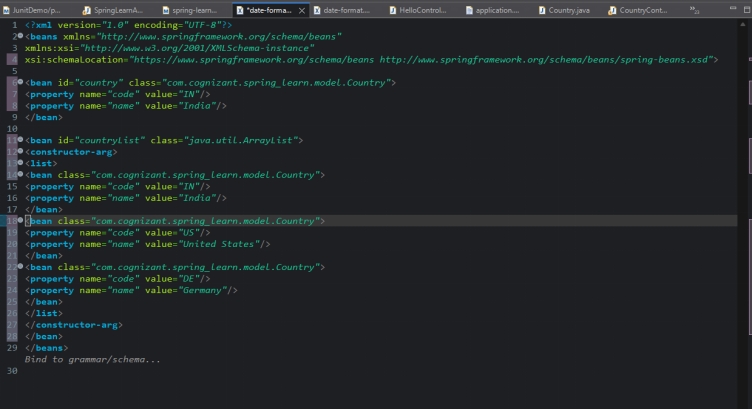
"code": "IN",

"name": "India"

}

**Step-by-Step Implementation**

**Update country.xml with Country List**

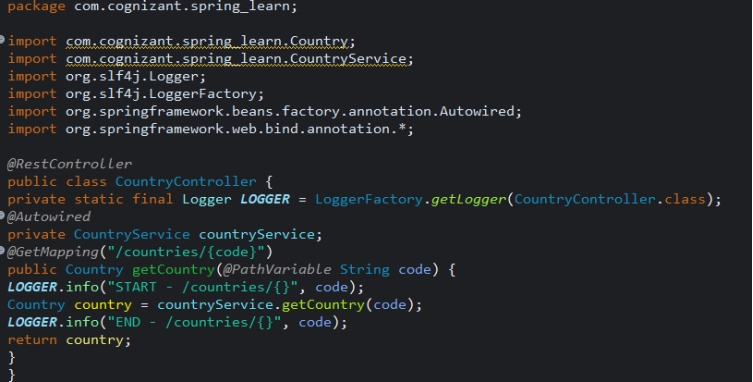


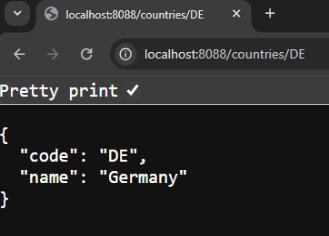
**Create CountryService Class**

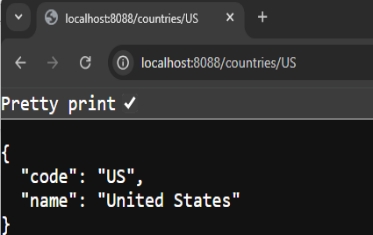
CountryService.java



**Update CountryController**







**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:**

<!-- Spring Boot DevTools -->

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- Spring Boot Starter Test -->

<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>

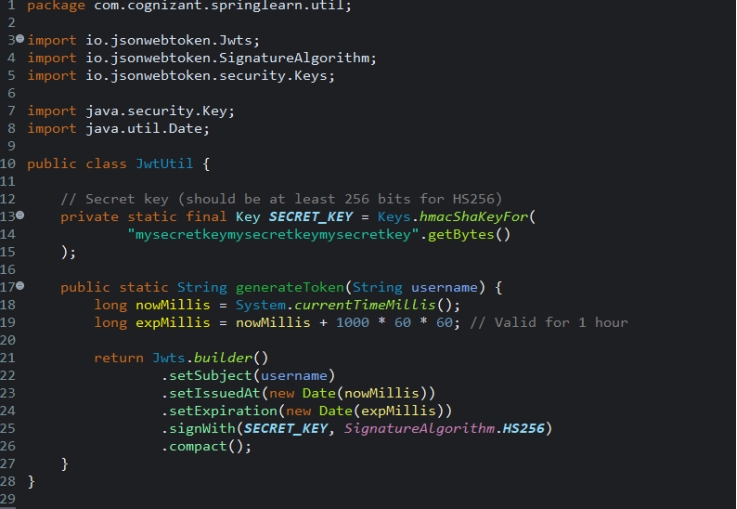
<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

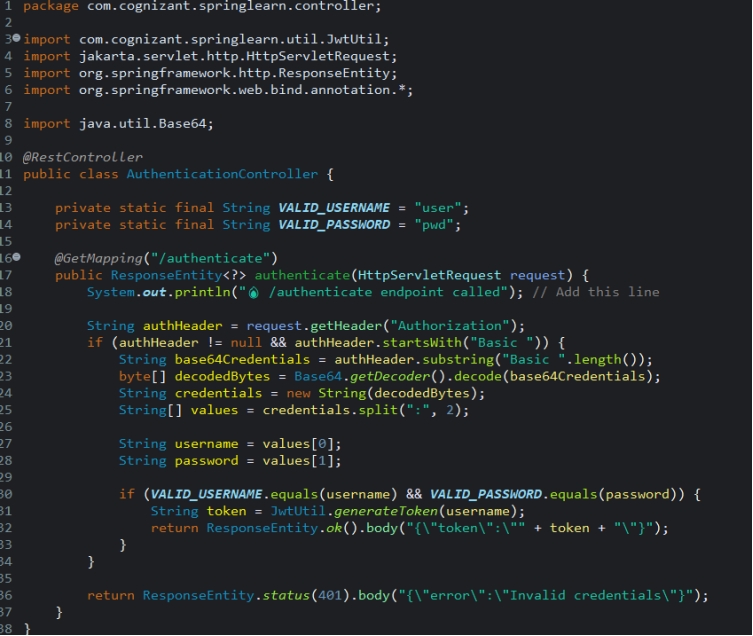
<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

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

**Step-3: Create Authentication Controller**



**Step-4: Run Application and Test with curl**

Open your terminal and run : curl -s -u user:pwd <http://localhost:8095/authenticate>

