Spring REST using Spring Boot 3

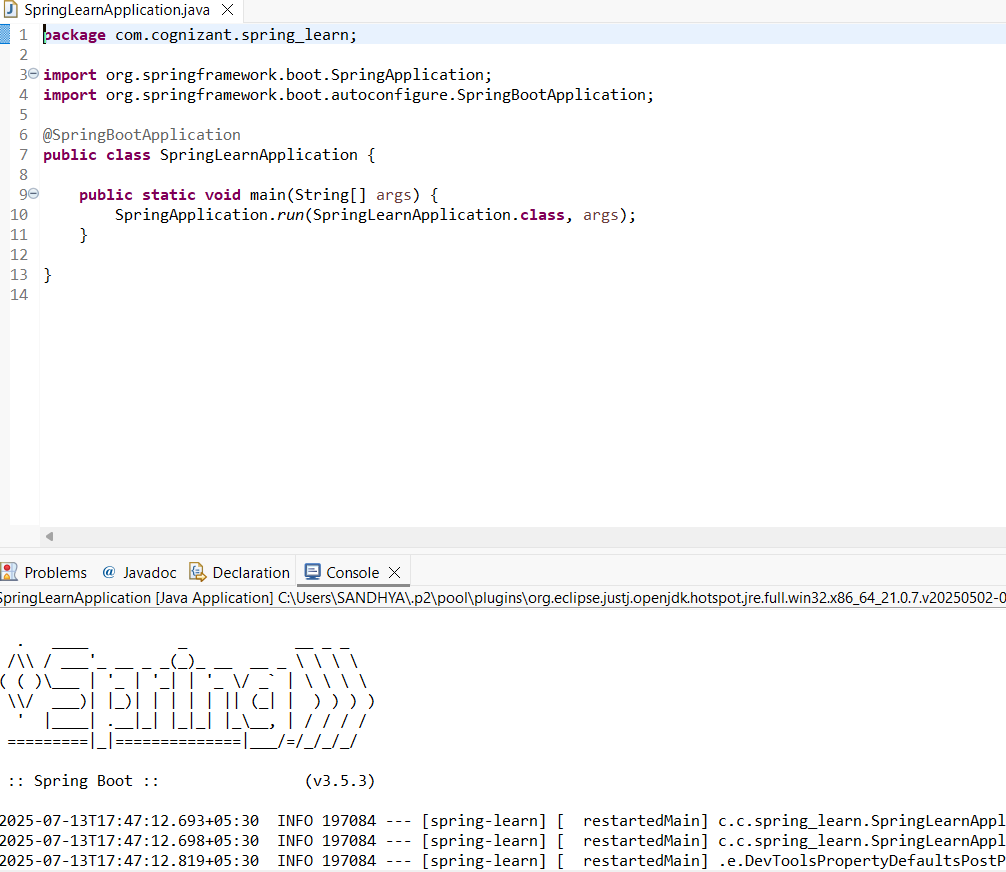
**1.Create a Spring Web Project using Maven**

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.

**OUTPUT:**

****

**2.Spring REST using Spring Boot 3**

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

**Country.java**

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Country {

private String code;

private String name;

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

public Country() {

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

}

public String getCode() {

*LOGGER*.debug("Inside getCode()");

return code;

}

public void setCode(String code) {

*LOGGER*.debug("Inside setCode()");

this.code = code;

}

public String getName() {

*LOGGER*.debug("Inside getName()");

return name;

}

public void setName(String name) {

*LOGGER*.debug("Inside setName()");

this.name = name;

}

@Override

public String toString() {

return "Country{" +

"code='" + code + '\'' +

", name='" + name + '\'' +

'}';

}

}

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

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 main(String[] args) {

*displayCountry*();

}

public static void displayCountry() {

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

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

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

}

}

**LogBack.xml**

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>

**Country.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">

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

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

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

</bean>

</beans>

**Log4j.xml**

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

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration>

<appender name="console" class="org.apache.log4j.ConsoleAppender">

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%d [%t] %-5p %c - %m%n"/>

</layout>

</appender>

<root>

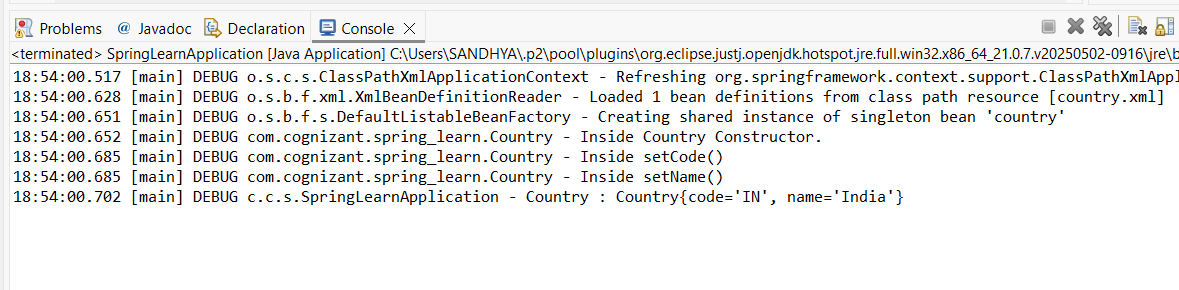
<priority value="debug"/>

<appender-ref ref="console"/>

</root>

</log4j:configuration>

**OUTPUT:**



---------------------------------

**Spring REST using Spring Boot 3**

**3.Hello World RESTful Web Service**

**HelloController.java**

package com.example.demo.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()");

*LOGGER*.debug("End: sayHello()");

return "Hello World!!";

}

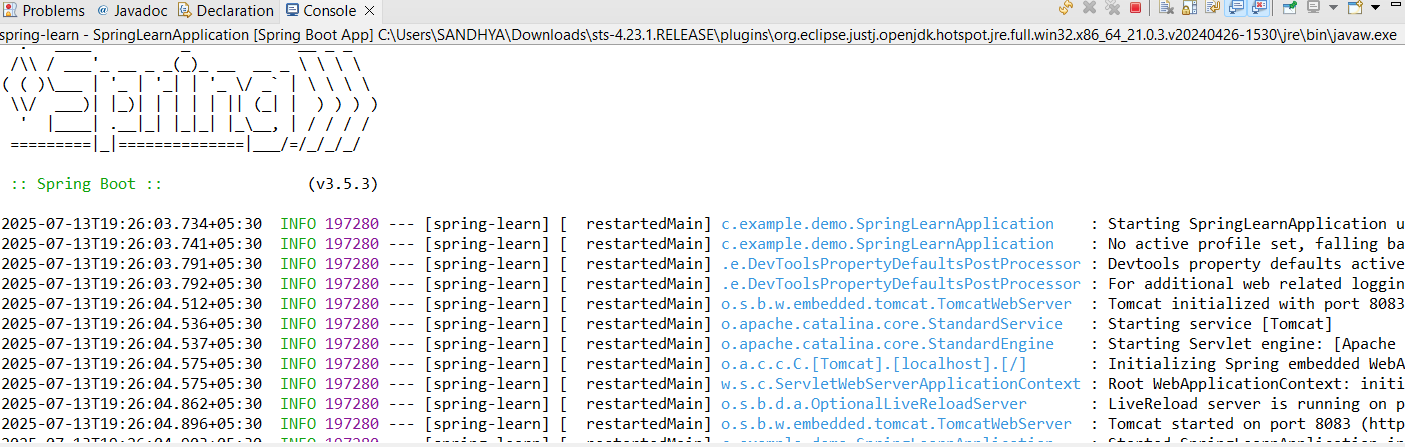
}

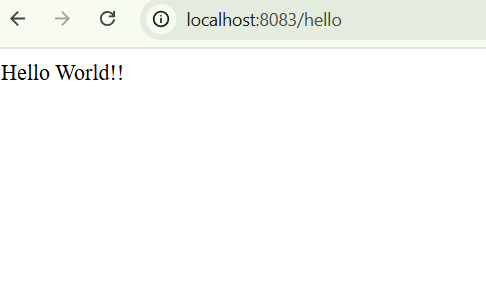
**POM.xml**

spring.application.name=spring-learn

server.port=8083

**RESULT:**

****



**4.REST – Get Country Based on Country Code**

Introduction

This document describes how to implement a Spring Boot RESTful web service that retrieves a specific country by its code from a list of countries defined in an XML configuration. The search should be case-insensitive, and the result should be returned in JSON format.

Objective

Create a REST endpoint:

* **URL**: /countries/{code}
* **HTTP Method**: GET
* **Path Variable**: code (country code, case-insensitive)
* **Controller**: com.cognizant.spring-learn.controller.CountryController
* **Service**: com.cognizant.spring-learn.service.CountryService.getCountry(String code)

Sample Request:

GET http://localhost:8083/countries/in

Sample Response:

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

Implementation Details

XML Configuration File (country.xml)

<**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.springlearn.model.Country">  
 <**property** name="code" value="IN"/>  
 <**property** name="name" value="India"/>  
 </**bean**>  
 <**bean** class="com.cognizant.springlearn.model.Country">  
 <**property** name="code" value="US"/>  
 <**property** name="name" value="United States"/>  
 </**bean**>  
 *<!-- Add more countries as needed -->*  
 </**list**>  
 </**constructor-arg**>  
 </**bean**>  
</**beans**>

Model Class: Country.java

**package com**.**cognizant**.**springlearn**.**model**;  
  
**public** **class** Country {  
 **private** String code;  
 **private** String name;  
  
 **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; }  
}

Service Class: CountryService.java

**package com**.**cognizant**.**springlearn**.**service**;  
  
**import** **com**.**cognizant**.**springlearn**.**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("country.xml");  
 List<Country> countries = (List<Country>) context.getBean("countryList");  
  
 **return** countries.stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElse(**null**);  
 }  
}

Controller Class: CountryController.java

**package com**.**cognizant**.**springlearn**.**controller**;  
  
**import** **com**.**cognizant**.**springlearn**.**model**.**Country**;  
**import** **com**.**cognizant**.**springlearn**.**service**.**CountryService**;  
**import** **org**.**springframework**.**beans**.**factory**.**annotation**.**Autowired**;  
**import** **org**.**springframework**.**web**.**bind**.**annotation**.\*;  
  
@RestController  
**public** **class** CountryController {  
  
 @Autowired  
 **private** CountryService countryService;  
  
 @GetMapping("/countries/{code}")  
 **public** Country getCountry(@PathVariable String code) {  
 **return** countryService.getCountry(code);  
 }  
}

Configuration: application.properties

server.port=8083

Behind the Scenes

* The @GetMapping("/countries/{code}") maps HTTP GET requests to the method getCountry() in the controller.
* The @PathVariable annotation extracts the country code from the URL.
* The service method retrieves the list of countries from country.xml.
* It performs a case-insensitive match using equalsIgnoreCase() in a stream filter.
* On finding the match, it returns the corresponding Country object, which Spring converts into JSON.

Sample Test

Request

GET http://localhost:8083/countries/in

Response

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

Conclusion

This RESTful web service demonstrates how to extract a path variable from a URL and use it to perform a case-insensitive search within a list of pre-configured beans. It illustrates key Spring concepts such as @PathVariable, @GetMapping, service layer abstraction, and XML-based bean configuration.

By using lambda expressions and Spring Boot features, the service remains concise, clean, and easy to extend in future applications.

**5.REST – Country Web Service in Spring Boot**

Introduction

This document outlines the implementation of a simple RESTful web service in a Spring Boot application that returns details of a country — specifically India. The service loads a Country bean defined in an XML configuration and responds with the country’s information in JSON format. This serves as a foundational exercise in understanding Spring MVC REST controllers, bean configuration, and JSON serialization.

Objective

Create a REST endpoint:

* **URL**: /country
* **Controller**: com.cognizant.spring-learn.controller.CountryController
* **Bean Source**: XML file (country.xml)
* **HTTP Method**: GET
* **Expected Response**:

Project Setup

Dependencies

* Spring Boot Starter Web
* Spring Context (for XML bean loading)
* Jackson (auto-included with Spring Boot for JSON conversion)

Component Implementation

Model Class

*// Country.java*  
**package com**.**cognizant**.**springlearn**.**model**;  
  
**public** **class** Country {  
 **private** String code;  
 **private** String name;  
  
 **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; }  
}

XML Bean Configuration

*<!-- country.xml -->*  
<**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="in" class="com.cognizant.springlearn.model.Country">  
 <**property** name="code" value="IN"/>  
 <**property** name="name" value="India"/>  
 </**bean**>  
</**beans**>

Spring Boot Main Class

*// 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);  
 }  
}

Controller Class

*// CountryController.java*  
**package com**.**cognizant**.**springlearn**.**controller**;  
  
**import** **com**.**cognizant**.**springlearn**.**model**.**Country**;  
**import** **org**.**springframework**.**context**.**ApplicationContext**;  
**import** **org**.**springframework**.**context**.**support**.**ClassPathXmlApplicationContext**;  
**import** **org**.**springframework**.**web**.**bind**.**annotation**.\*;  
  
@RestController  
**public** **class** CountryController {  
  
 @RequestMapping("/country")  
 **public** Country getCountryIndia() {  
 ApplicationContext context = **new** ClassPathXmlApplicationContext("country.xml");  
 Country country = (Country) context.getBean("in");  
 **return** country;  
 }  
}

Configuration File

# application.properties  
server.port=8083

Behind the Scenes

What Happens in the Controller Method?

* The @RequestMapping("/country") maps incoming GET requests to the method getCountryIndia().
* Inside this method:
  + ClassPathXmlApplicationContext loads the country.xml.
  + The Country bean with ID in is retrieved and returned.
* Spring’s @RestController automatically converts the returned Java object to JSON.

How is the Bean Converted to JSON?

* Spring Boot uses **Jackson** for object-to-JSON conversion.
* Jackson introspects the Country object (via getters/setters) and serializes it to a valid JSON string.
* No need for manual conversion — Spring does it automatically behind the scenes.

Testing the Service

Request

GET http://localhost:8083/country

Response

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

Verifying in Tools

Developer Tools (Browser Network Tab)

* **Request URL**: http://localhost:8083/country
* **Request Method**: GET
* **Status Code**: 200 OK
* **Content-Type**: application/json
* **Response Payload**:

Postman Headers Tab

When testing in Postman:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Content-Length | 31 |
| Date | [Current Timestamp] |
| Server | Apache Tomcat/Embedded |

Conclusion

This project demonstrates a simple yet effective use of Spring Boot to build a RESTful service that loads a Java bean from an XML configuration and returns it as a JSON response. The use of annotations like @RestController and @RequestMapping, combined with Spring’s integration with Jackson, makes it seamless to build and test REST APIs.

**6.JWT Authentication in Spring Boot – Step-by-Step Implementation**

**Step 1: Create a Spring Boot Project**

Use Spring Initializr or your IDE (like Spring Tool Suite or IntelliJ) to create a Maven-based Spring Boot project with the following dependencies:

* spring-boot-starter-web
* spring-boot-starter-security
* jjwt-api
* jjwt-impl
* jjwt-jackson

**Sample pom.xml Dependencies**

xml

CopyEdit

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<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>

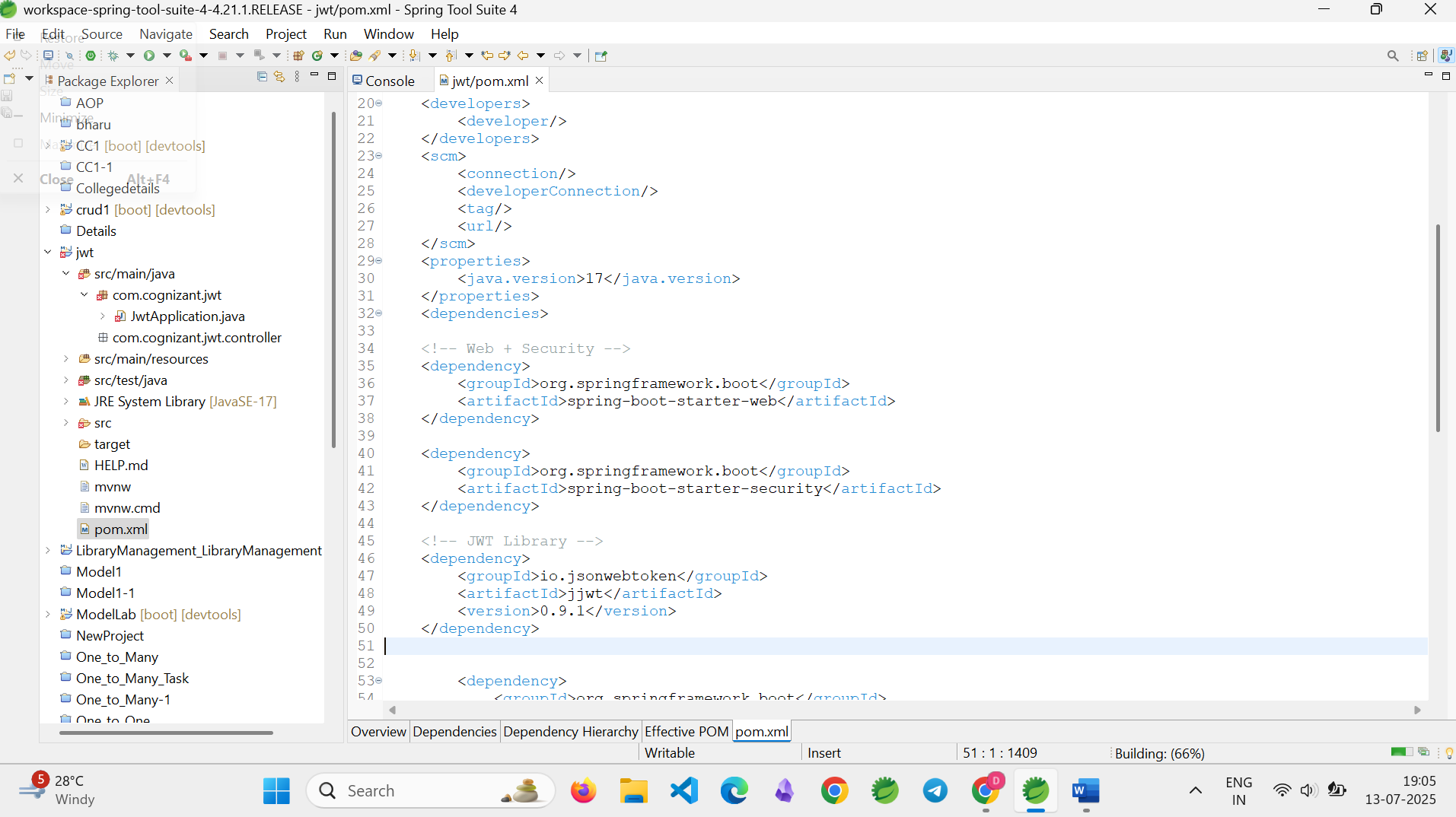
<artifactId>jjwt-jackson</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

</dependencies>



**Step 2: Configure Application Port**

Open src/main/resources/application.properties and add the following line:

server.port=8091

This sets the application to run on port 8091.

**Step 3: Create the Main Application Class**

Create a main class named JwtApplication.java in the base package:

@SpringBootApplication

public class JwtApplication {

public static void main(String[] args) {

SpringApplication.run(JwtApplication.class, args);

}

}

**Step 4: Create the JWT Service**

Create a class JwtService.java under com.cognizant.jwt.service.

@Service

public class JwtService {

private final String secretKey = "myverysecuresecretkey123456789012";

public String[] extractCredentials(String authHeader) {

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

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

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

String credentials = new String(decoded);

return credentials.split(":", 2);

}

throw new RuntimeException("Missing or invalid Authorization header");

}

public String generateToken(String username) {

long now = System.currentTimeMillis();

long expiry = now + (10 \* 60 \* 1000);

Key key = Keys.hmacShaKeyFor(secretKey.getBytes());

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date(now))

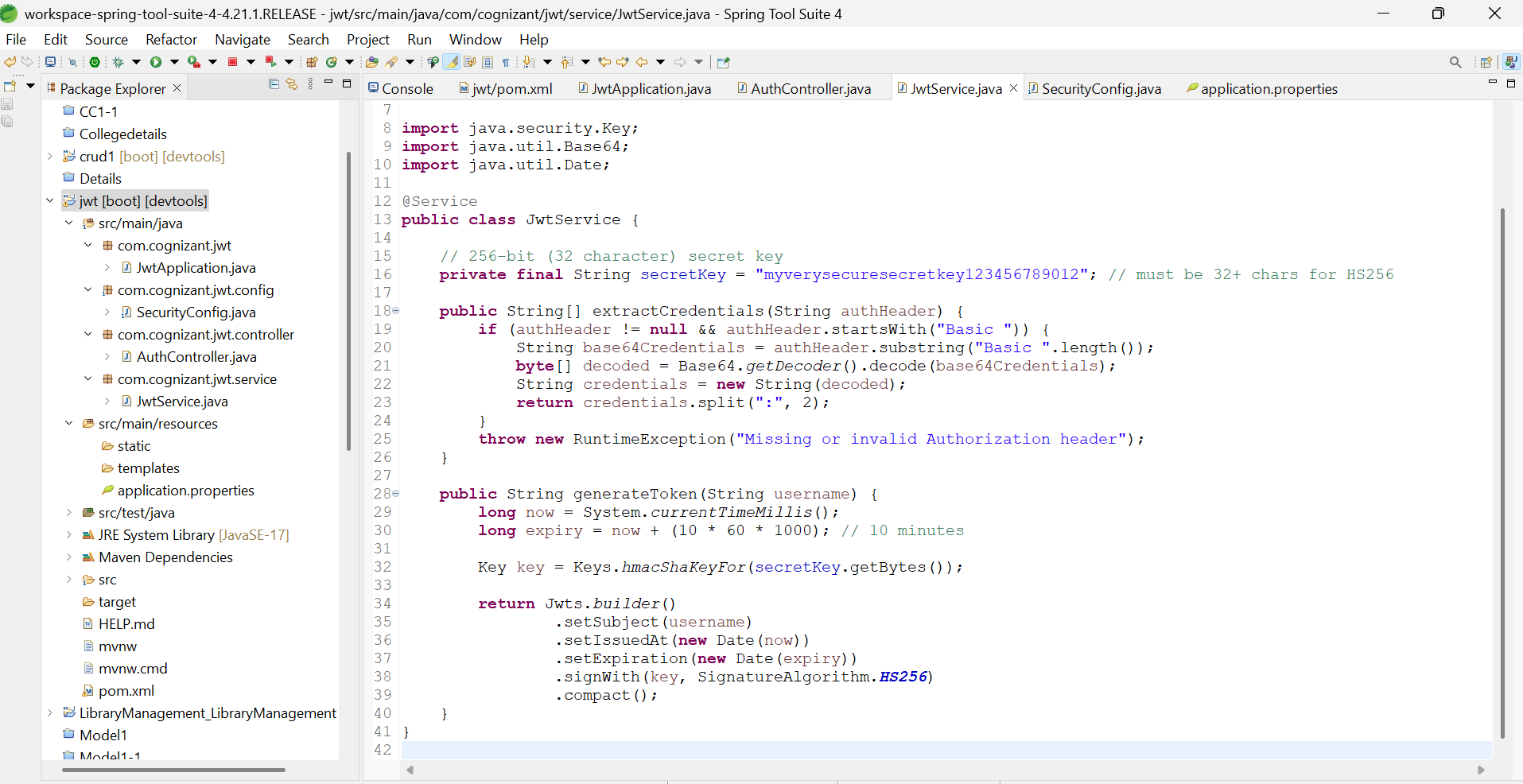
.setExpiration(new Date(expiry))

.signWith(key, SignatureAlgorithm.HS256)

.compact();

}

}



**Step 5: Create the Authentication Controller**

Create a class named AuthController.java under com.cognizant.jwt.controller.

@RestController

public class AuthController {

@Autowired

private JwtService jwtService;

@GetMapping("/authenticate")

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

try {

String[] creds = jwtService.extractCredentials(authHeader);

String username = creds[0];

String password = creds[1];

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

String token = jwtService.generateToken(username);

return ResponseEntity.ok(Collections.singletonMap("token", token));

} else {

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

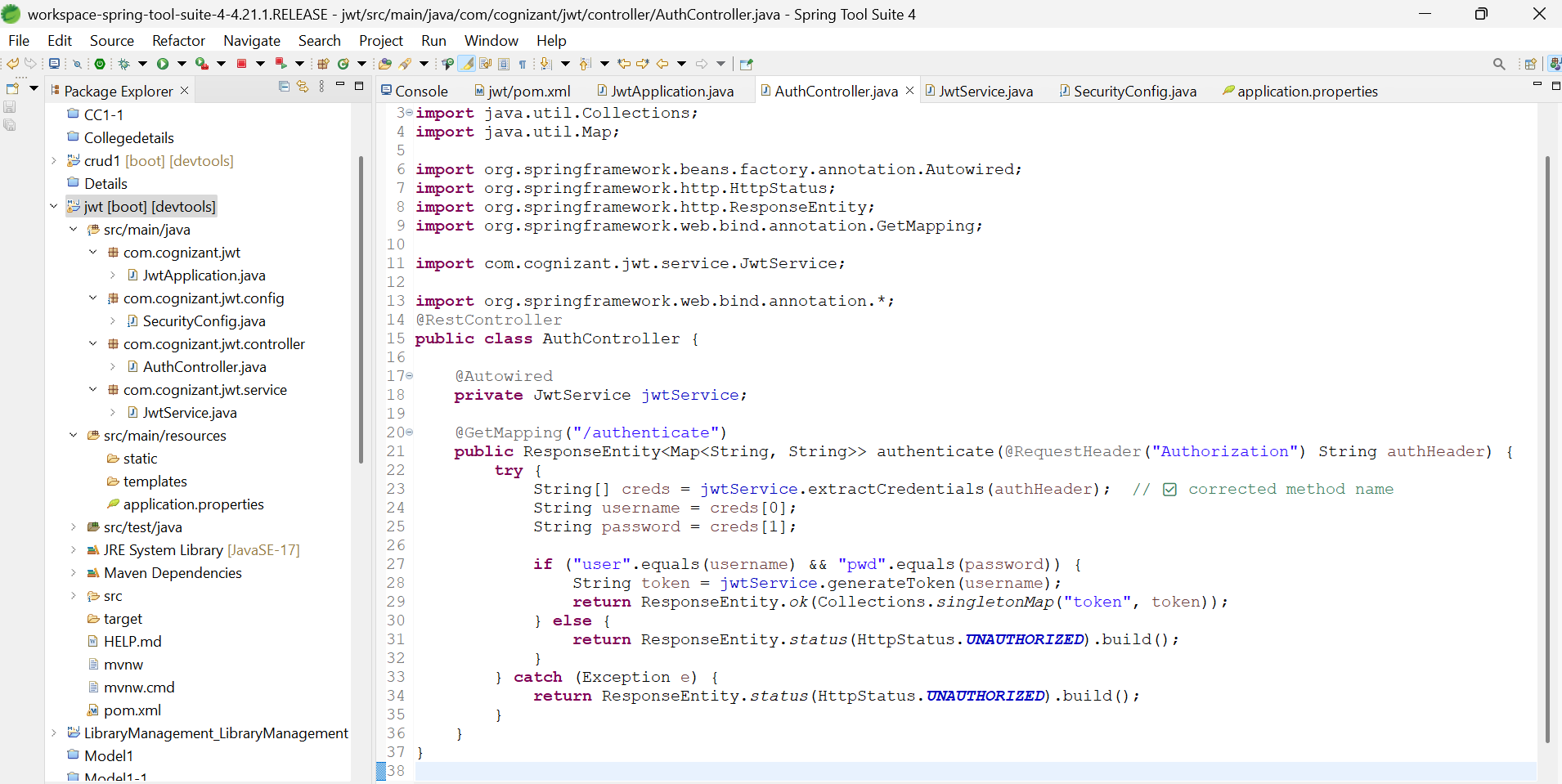
}

} catch (Exception e) {

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

}

}

}

**Step 6: Configure Spring Security**

Create a configuration class SecurityConfig.java under com.cognizant.jwt.config.

@Configuration

@EnableWebSecurity

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();

}

@Bean

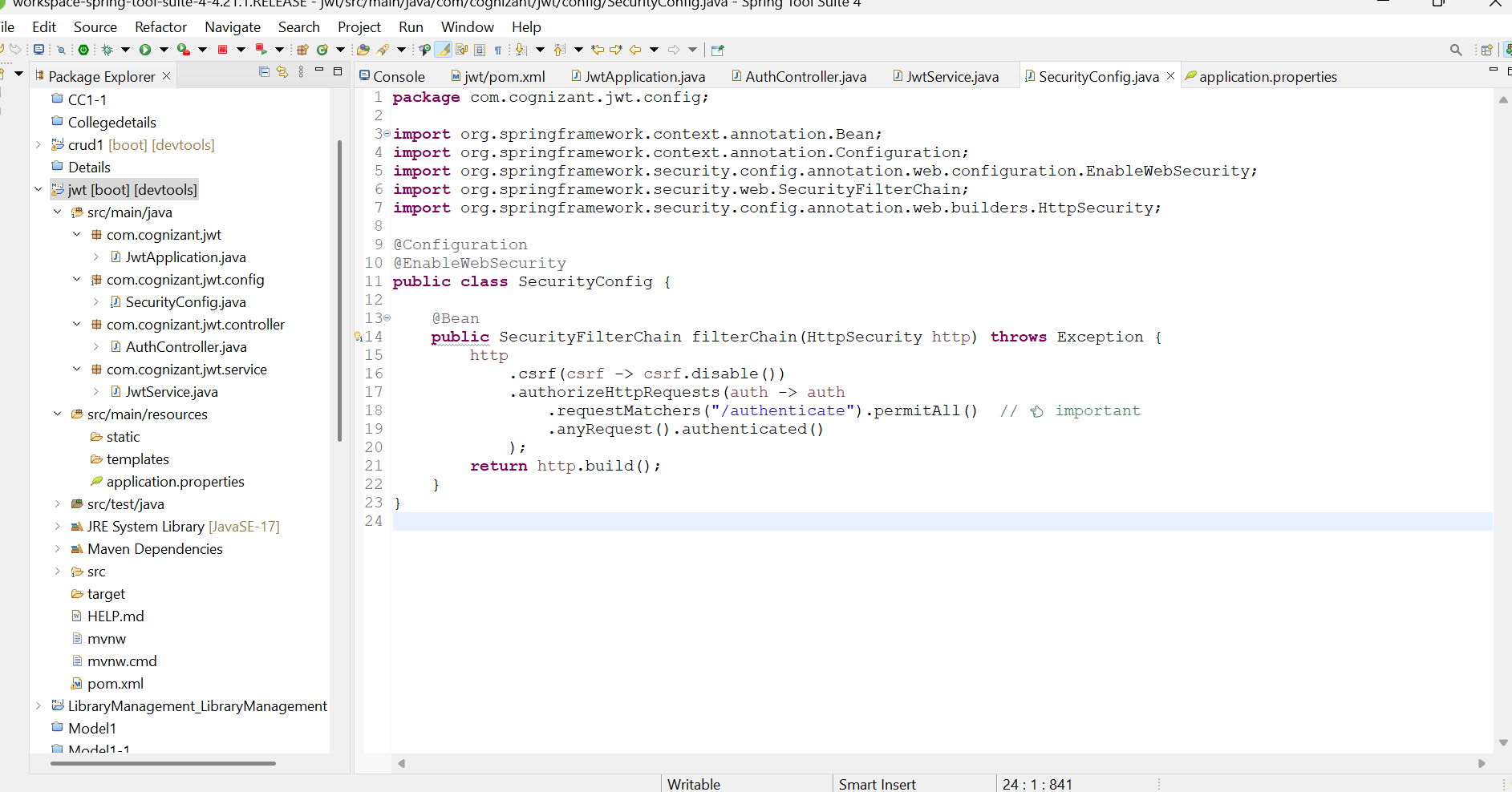
public UserDetailsService userDetailsService() {

return username -> null;

}

}

This disables CSRF, allows access to the /authenticate endpoint without authentication, and disables Spring Boot’s default user authentication.



**Step 7: Test the JWT Authentication Endpoint**

Use curl or Postman to test the /authenticate endpoint.

curl -u user:pwd http://localhost:8091/authenticate

Expected Output:

{

"token": "eyJhbGciOiJIUzI1NiJ9..."

}

Postman:

* Method: GET
* URL: <http://localhost:8091/authenticate>
* Authorization tab: Basic Auth (username = user, password = pwd)

If password wrong

401 error Unauthorized

