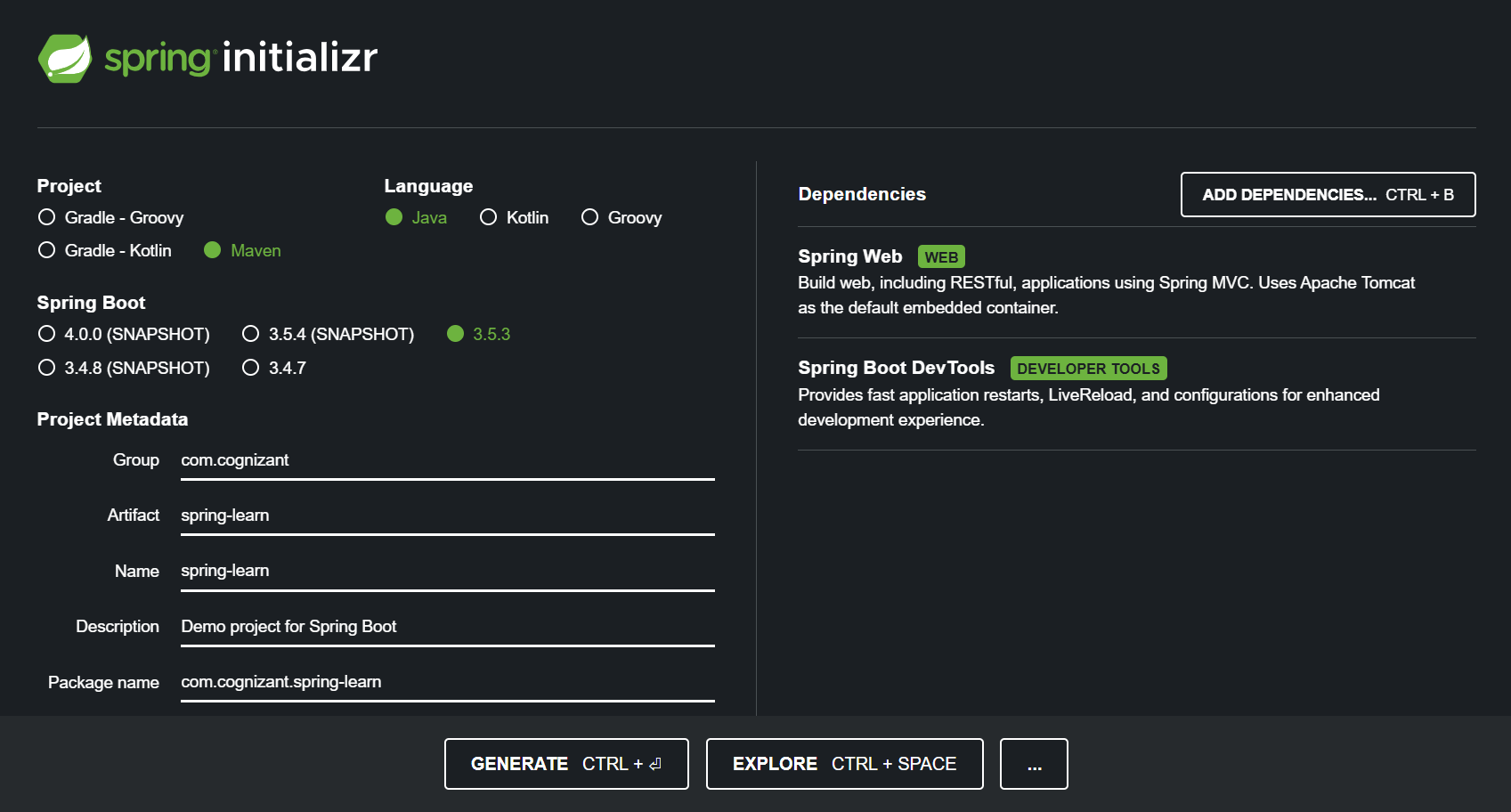
Week-3 Hands-on Spring REST using Spring Boot 3:

1. Create a Spring Web Project using Maven:



1. Spring Core – Load Country from Spring Configuration XML:

**·** Configuring Spring XML configuration named country.xml.

**·** Creating a bean tag in spring configuration for country and set the property and values

File name: 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>

Creating 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 , method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country

File name: Country.java

package com.cognizant.spring\_learn;

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");

this.code = code;

}

public String getName() {

LOGGER.debug("Getting name");

return name;

}

public void setName(String name) {

LOGGER.debug("Setting name");

this.name = name;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

* 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

File 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 displayCountry() {

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

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

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

}

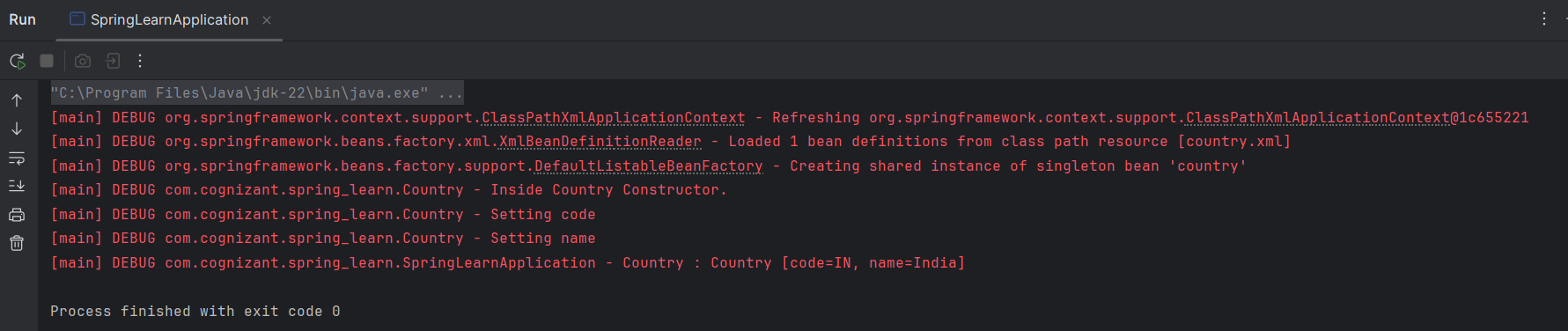
public static void main(String[] args) {

displayCountry();

}

}

OUTPUT:



3. Hello World RESTful Web Service:

* Created the controller class for implementing the rest service for returning a text “Hello World”

File name: HelloController.java

package com.cognizant.spring\_learn.Contoller;

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 response = "Hello World!!";

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

return response;

}

}

File name: SpringLearnApplication.java

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) {

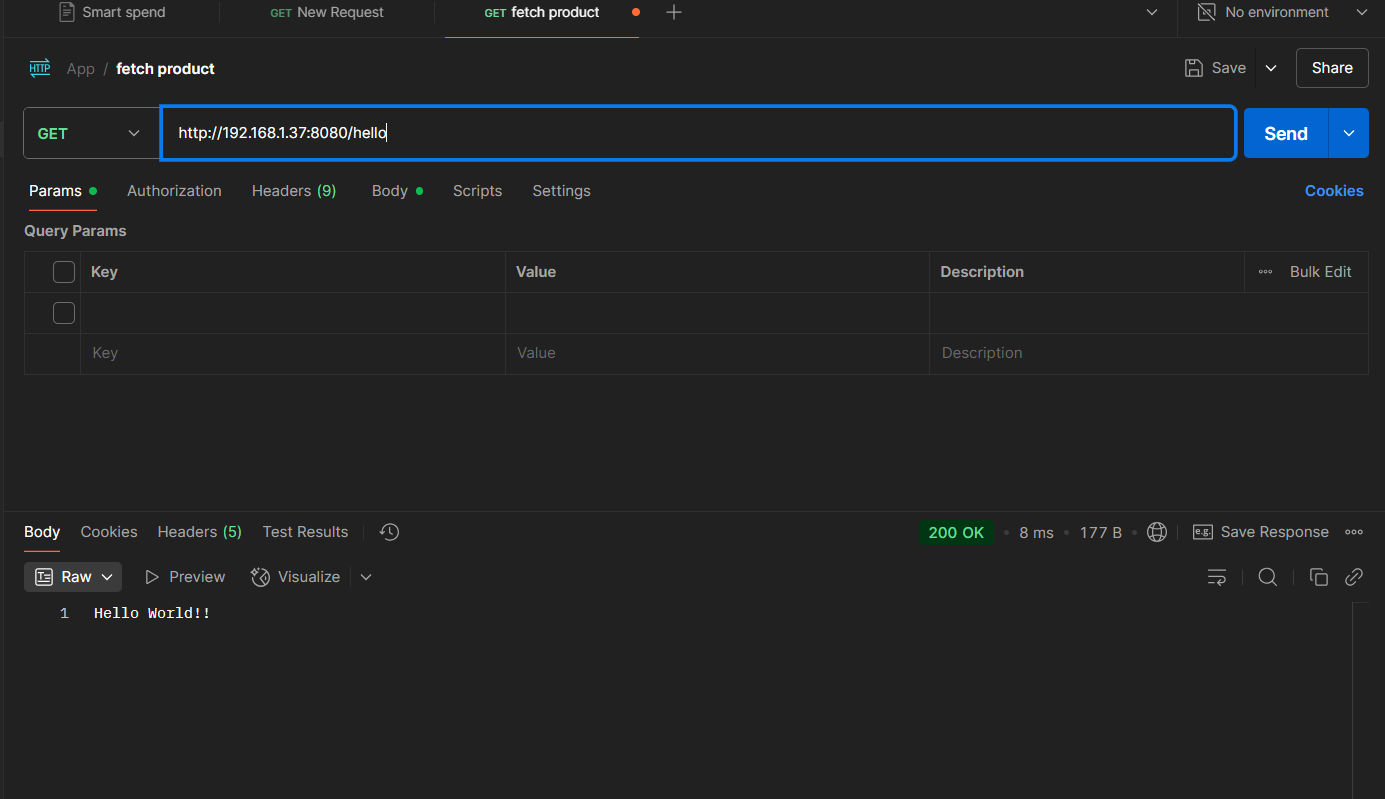
SpringApplication.run(SpringLearnApplication.class, args);

}

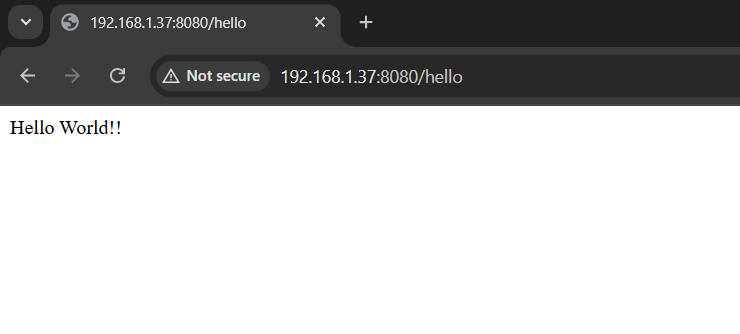
}

OUTPUT:

Testing the Api service with post man



Testing the Api service in chrome browser



1. REST - Country Web Service:

* Created controller class for country to handle the service

File name: CountryController.java

package com.cognizant.spring\_learn.Contoller;

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

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

LOGGER.debug("END getCountryIndia()");

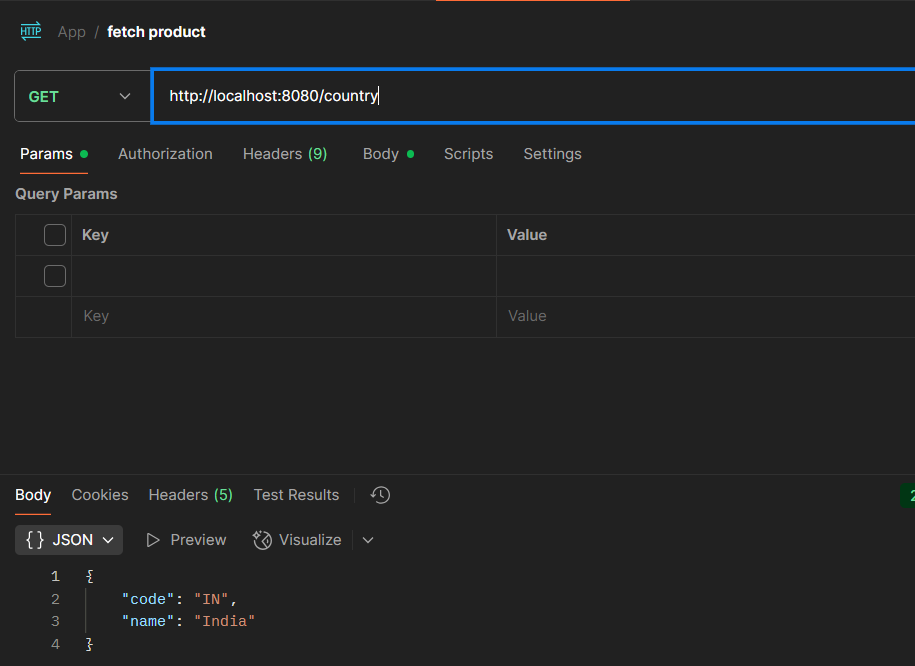
return country;

}

}

OUTPUT:

Testing the country web service using postman



1. REST - Get country based on country code:

* Created a method inside the controller class with the @GetMapping annotation

File name: CountryController.java

package com.cognizant.spring\_learn.Contoller;

import com.cognizant.spring\_learn.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.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

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

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

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

@RestController

public class CountryController {

@Autowired

private CountryService countryService;

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

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

public Country getCountry(@PathVariable String code) {

LOGGER.info("START getCountry - code: {}", code);

Country country = countryService.getCountry(code);

LOGGER.info("END getCountry");

return country;

}

@RequestMapping("/country")

public Country getCountryIndia() {

LOGGER.debug("START getCountryIndia()");

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

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

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

LOGGER.debug("END getCountryIndia()");

return country;

}

}

* Created a service class for handling the service. The getCountry() service method gets the country.xml and iterate the countries list

File name: CountryService.java

package com.cognizant.spring\_learn.Service;

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.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.info("START getCountry with code: {}", code);

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

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

return countryList.stream()

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

.findFirst()

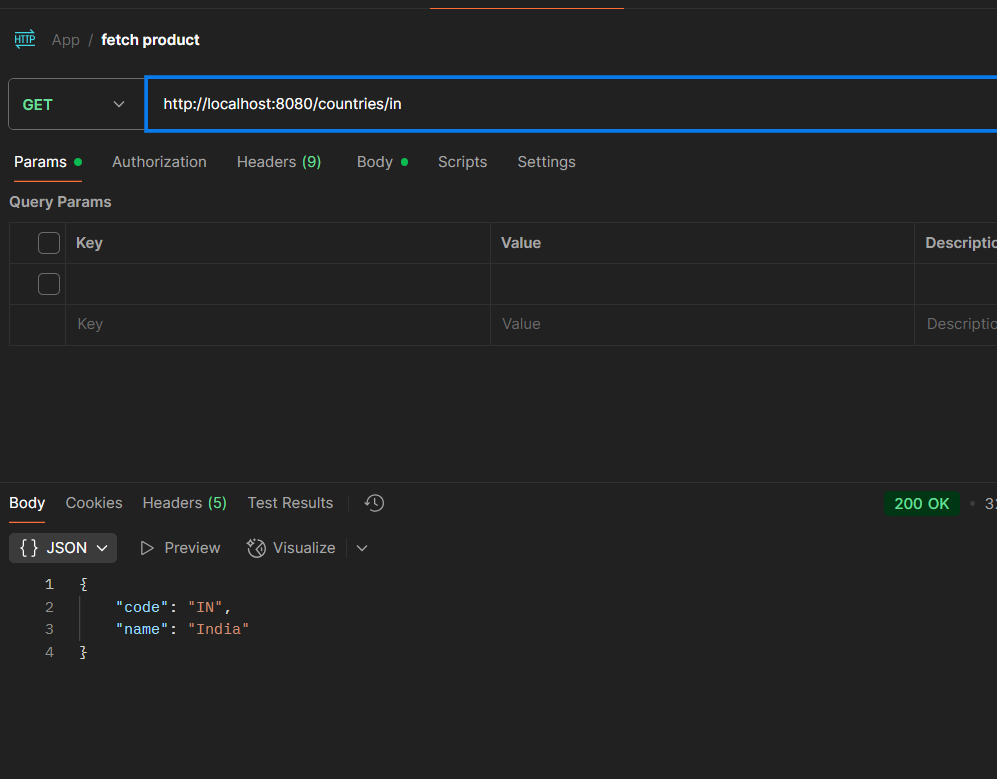
.orElse(null);

}

}

OUTPUT:

Tested the get country by code service using postman



1. Create authentication service that returns JWT:

* Creating the controller class for Authentication

File name: AuthenticationController.java

package com.cognizant.spring\_learn.Contoller;

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

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

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

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.AuthenticationException;

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

@RestController

public class AuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtUtil jwtUtil;

@RequestMapping(value = "/authenticate", method = RequestMethod.GET)

public AuthenticationResponse generateToken(@RequestHeader("Authorization") String authHeader) {

try {

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

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

String decodedCredentials = new String(decodedBytes);

String[] parts = decodedCredentials.split(":");

String username = parts[0];

String password = parts[1];

authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

String token = jwtUtil.generateToken(username);

return new AuthenticationResponse(token);

} catch (AuthenticationException e) {

throw new RuntimeException("Invalid Credentials");

}

}

}

File name: SecurityConfig.java

package com.cognizant.spring\_learn.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.crypto.password.NoOpPasswordEncoder;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

@Bean

public UserDetailsService userDetailsService() {

return new InMemoryUserDetailsManager(

User.withUsername("user")

.password("pwd")

.roles("USER")

.build()

);

}

@Bean

public AuthenticationManager authenticationManager(HttpSecurity http, UserDetailsService userDetailsService) throws Exception {

return http.getSharedObject(AuthenticationManagerBuilder.class)

.userDetailsService(userDetailsService)

.passwordEncoder(passwordEncoder())

.and()

.build();

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests().anyRequest().authenticated()

.and().httpBasic();

return http.build();

}

@SuppressWarnings("deprecation")

@Bean

public static NoOpPasswordEncoder passwordEncoder() {

return (NoOpPasswordEncoder) NoOpPasswordEncoder.getInstance();

}

}

* Generating token based on user retrieval

File name: JwtUtil.java

package com.cognizant.spring\_learn.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;

import java.security.Key;

import java.util.Date;

@Component

public class JwtUtil {

private final String SECRET\_KEY = "karthikeyan-your-custom-secret-key-123456"; // Use min 256-bit key (32+ characters)

public String generateToken(String username) {

Key key = Keys.hmacShaKeyFor(SECRET\_KEY.getBytes(StandardCharsets.UTF\_8));

return Jwts.builder()

.setSubject(username)

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

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10)) // 10 hours

.signWith(key, SignatureAlgorithm.HS256)

.compact();

}

}

OUTPUT:

