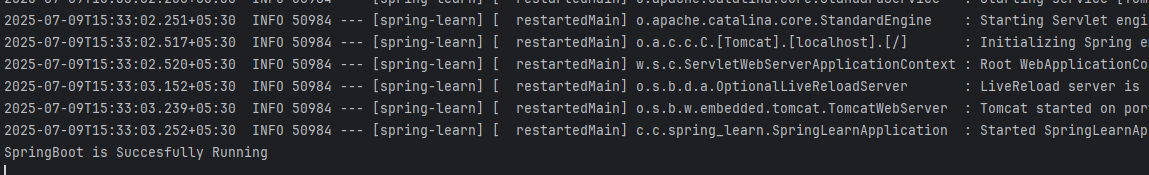
**Hands on 1**

**Create a Spring Web Project using Maven** :

**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);  
 System.*out*.println("SpringBoot is Succesfully Running");  
 }  
  
}

**Output:**

  
**Hands on 4**

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

**Country.java:**

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("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 + "]";  
 }  
}

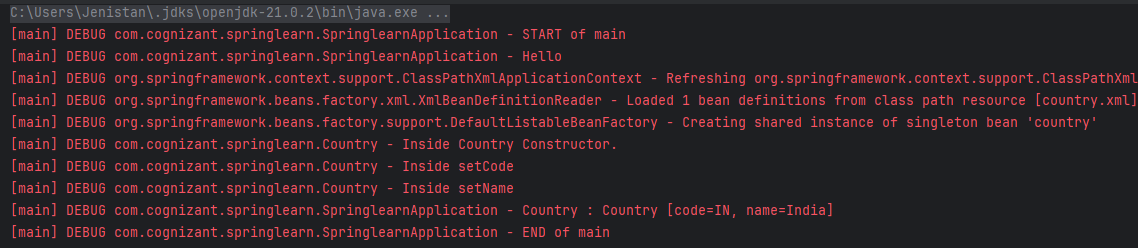
**Country.xml:**

<?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  
 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>

**SpringlearnApplication.java:**

public class SpringlearnApplication {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(SpringlearnApplication.class);  
  
 public static void main(String[] args) {  
 *LOGGER*.debug("START of main");  
 *LOGGER*.debug("Hello");  
 *displayCountry*();  
 *LOGGER*.debug("END of main");  
 }  
 public static void displayCountry() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 Country country = context.getBean("country", Country.class);  
 *LOGGER*.debug("Country : {}", country.toString());  
 }  
}

**Output:**

****

**bean Tag**

* Declares a bean (i.e., an object) that will be managed by the Spring IoC container.
* Spring creates and manages this object during application startup or when requested.

<bean id="country" class="com.cognizant.springlearn.Country">

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

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

</bean>

**id Attribute**

* A unique name used to refer to the bean inside Spring’s context.
* You use this name with context.getBean("id") to retrieve the bean.

**class Attribute**

* Fully qualified Java class name for the bean to instantiate.
* Spring uses reflection to create an object of this class.

**property Tag**

* Sets the values of fields (properties) inside the bean after it's instantiated.

**name Attribute:**

* Specifies the Java property name (matching the setter method name).
* If you use <property name="code">, Spring looks for setCode() method in the class.

**value Attribute :**

* Specifies the literal value to inject into the field.

**ApplicationContext:**

* Central interface in Spring for providing configuration information to the application.
* Manages beans (their lifecycle, dependencies, configuration, etc.).
* More advanced than BeanFactory.

**ClassPathXmlApplicationContext:**

* Concrete class that implements ApplicationContext.
* It loads configuration XML from the classpath, parses the bean definitions, and initializes the beans.

**Hello World RESTful Web Service** **:**  
**Code:**

**Country.java:**

public class Country {  
 private String code;  
 private String name;  
 public Country() {}  
 public Country(String code, String name) {  
 this.code = code;  
 this.name = 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; }  
}

**HelloController.java:**

@RestController  
public class HelloController {  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(HelloController.class);  
  
 @GetMapping("/hello")  
 public String sayHello() {  
 *LOGGER*.info("START: sayHello()");  
 *LOGGER*.info("END: sayHello()");  
 return "Hello World!!";  
 }  
}

**CountryController.java:**

@RestController  
public class CountryController {  
  
 @Autowired  
 private CountryService countryService;  
  
 @RequestMapping("/country")  
 public Country getCountryIndia() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return (Country) context.getBean("in");  
 }  
  
 @GetMapping("/countries")  
 public List<Country> getAllCountries() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return (List<Country>) context.getBean("countryList");  
 }  
  
 @GetMapping("/countries/{code}")  
 public Country getCountry(@PathVariable String code) throws CountryNotFoundException {  
 return countryService.getCountry(code);  
 }  
}

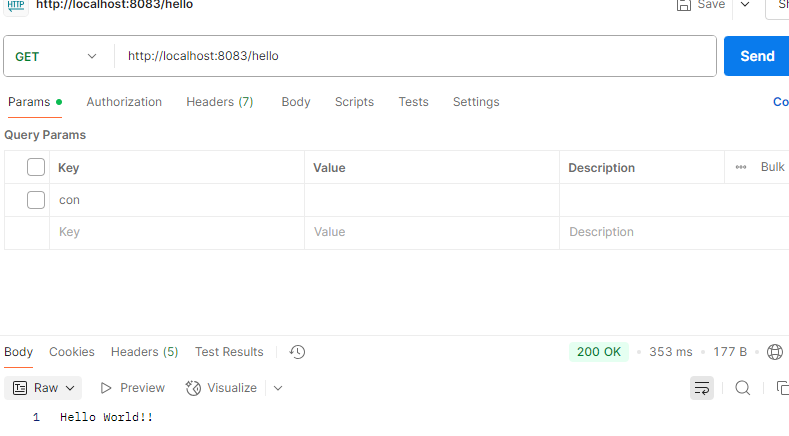
**CountryService.java:**

@Service  
public class CountryService {  
 public Country getCountry(String code) throws CountryNotFoundException {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 List<Country> countries = (List<Country>) context.getBean("countryList");  
  
 return countries.stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElseThrow(() -> new CountryNotFoundException("Country not found: " + code));  
 }  
}

**SpringlearnApplication.java:**

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

**Output:**

****

**REST - Country Web Service** **:**

**Code:**

**Country.java:**

public class Country {  
 private String code;  
 private String name;  
 public Country() {}  
 public Country(String code, String name) {  
 this.code = code;  
 this.name = 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; }  
}

**CountryController.java:**

@RestController  
public class CountryController {  
  
 @Autowired  
 private CountryService countryService;  
  
 @RequestMapping("/country")  
 public Country getCountryIndia() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return (Country) context.getBean("in");  
 }  
  
 @GetMapping("/countries")  
 public List<Country> getAllCountries() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return (List<Country>) context.getBean("countryList");  
 }  
  
 @GetMapping("/countries/{code}")  
 public Country getCountry(@PathVariable String code) throws CountryNotFoundException {  
 return countryService.getCountry(code);  
 }  
}

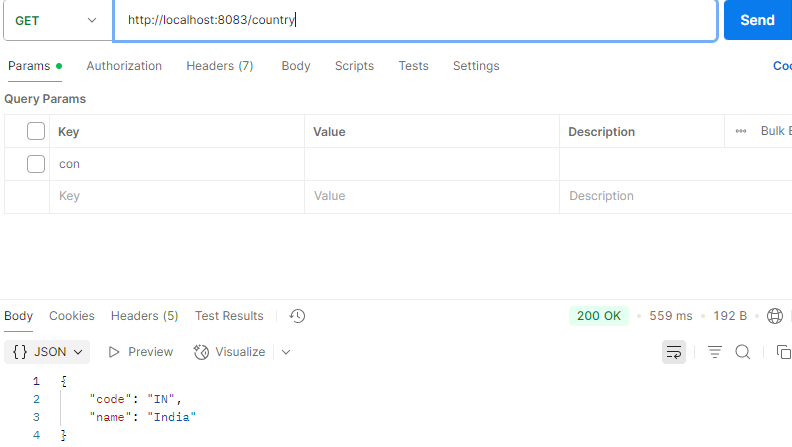
**CountryService.java:**

@Service  
public class CountryService {  
 public Country getCountry(String code) throws CountryNotFoundException {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 List<Country> countries = (List<Country>) context.getBean("countryList");  
  
 return countries.stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElseThrow(() -> new CountryNotFoundException("Country not found: " + code));  
 }  
}

**SpringlearnApplication.java:**

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

**Output:**

****

**REST - Get country based on country code** **:**

**Code:**

**Country.java:**

public class Country {  
 private String code;  
 private String name;  
 public Country() {}  
 public Country(String code, String name) {  
 this.code = code;  
 this.name = 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; }  
}

**CountryController.java:**

@RestController  
public class CountryController {  
  
 @Autowired  
 private CountryService countryService;  
  
 @RequestMapping("/country")  
 public Country getCountryIndia() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return (Country) context.getBean("in");  
 }  
  
 @GetMapping("/countries")  
 public List<Country> getAllCountries() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return (List<Country>) context.getBean("countryList");  
 }  
  
 @GetMapping("/countries/{code}")  
 public Country getCountry(@PathVariable String code) throws CountryNotFoundException {  
 return countryService.getCountry(code);  
 }  
}

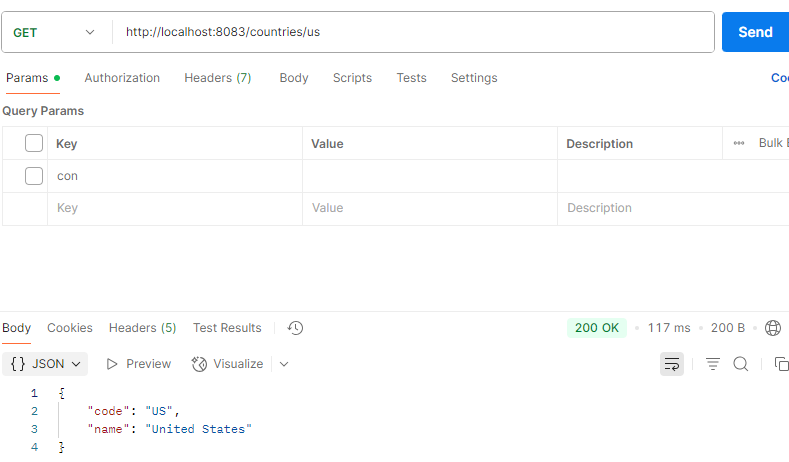
**CountryService.java:**

@Service  
public class CountryService {  
 public Country getCountry(String code) throws CountryNotFoundException {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 List<Country> countries = (List<Country>) context.getBean("countryList");  
  
 return countries.stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElseThrow(() -> new CountryNotFoundException("Country not found: " + code));  
 }  
}

**SpringlearnApplication.java:**

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

**Output:**

****

**JWT HandsOn:**

**Create authentication service that returns JWT** :

**Code:**

**JwtService.java:**

@Service  
public class JwtService {  
  
 private final String secretKey = "mySecretKey12345"; // Use env var in prod  
  
 public String generateTokenFromAuthHeader(String authHeader) {  
 if (authHeader == null || !authHeader.startsWith("Basic ")) {  
 throw new RuntimeException("Invalid Authorization header");  
 }  
  
 String base64Credentials = authHeader.substring("Basic ".length());  
 String decoded = new String(Base64.*getDecoder*().decode(base64Credentials), StandardCharsets.*UTF\_8*);  
 String[] credentials = decoded.split(":", 2);  
  
 String username = credentials[0];  
 String password = credentials[1];  
  
 if (!"user".equals(username) || !"pwd".equals(password)) {  
 throw new RuntimeException("Invalid username or password");  
 }  
  
 return generateToken(username);  
 }  
  
 private String generateToken(String username) {  
 return Jwts.*builder*()  
 .setSubject(username)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + 600000)) // 10 min  
 .signWith(SignatureAlgorithm.*HS256*, secretKey)  
 .compact();  
 }  
}

**AuthController.java:**

@RestController  
public class AuthController {  
  
 @Autowired  
 private JwtUtil jwtUtil;  
  
 @GetMapping("/authenticate")  
 public ResponseEntity<String> authenticate(HttpServletRequest request) {  
 try {  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader == null || !authHeader.startsWith("Basic ")) {  
 return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*).body("{\"error\": \"Missing Authorization header\"}");  
 }  
  
 String base64Credentials = authHeader.substring("Basic ".length());  
 byte[] credDecoded = Base64.*getDecoder*().decode(base64Credentials);  
 String credentials = new String(credDecoded, StandardCharsets.*UTF\_8*);  
  
 String[] values = credentials.split(":", 2);  
 if (values.length != 2) {  
 return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*).body("{\"error\": \"Invalid credentials\"}");  
 }  
  
 String username = values[0];  
 String password = values[1];  
  
 if ("user".equals(username) && "pwd".equals(password)) {  
 String token = jwtUtil.generateToken(username);  
 return ResponseEntity.*ok*("{\"token\":\"" + token + "\"}");  
 } else {  
 return ResponseEntity.*status*(HttpStatus.*UNAUTHORIZED*).body("{\"error\": \"Invalid credentials\"}");  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*)  
 .body("{\"error\": \"Server error: " + e.getMessage() + "\"}");  
 }  
 }  
  
 @GetMapping("/countries")  
 public ResponseEntity<String> getCountries() {  
 return ResponseEntity.*ok*("[{\"code\":\"US\",\"name\":\"United States\"},{\"code\":\"IN\",\"name\":\"India\"}]");  
 }  
}

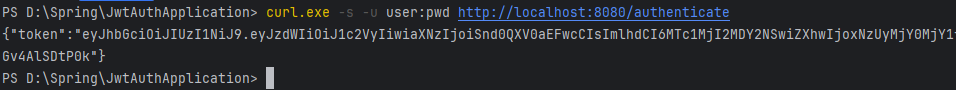
**SecurityConfig.java:**

@Configuration  
public class SecurityConfig {  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 // Define in-memory users  
 @Bean  
 public AuthenticationManager authManager(HttpSecurity http, PasswordEncoder encoder) throws Exception {  
 AuthenticationManagerBuilder authBuilder = http.getSharedObject(AuthenticationManagerBuilder.class);  
 authBuilder.inMemoryAuthentication()  
 .withUser("user").password(encoder.encode("pwd")).roles("USER")  
 .and()  
 .withUser("admin").password(encoder.encode("pwd")).roles("ADMIN");  
 return authBuilder.build();  
 }  
  
 // Define access rules  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http.csrf().disable()  
 .authorizeHttpRequests()  
 .requestMatchers("/authenticate").permitAll()  
 .requestMatchers("/countries").hasRole("USER")  
 .anyRequest().authenticated()  
 .and()  
 .httpBasic();  
 return http.build();  
 }  
}

**JwtUtil.java:**

@Component  
public class JwtUtil {  
 private final Key secretKey = Keys.*hmacShaKeyFor*("supersecretkeysupersecretkey123!".getBytes()); // min 32 bytes  
  
 public String generateToken(String username) {  
 return Jwts.*builder*()  
 .setSubject(username)  
 .setIssuer("JwtAuthApp")  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + 3600000)) // 1 hr  
 .signWith(secretKey, SignatureAlgorithm.*HS256*)  
 .compact();  
 }  
}

**Output:**

****