Spring Boot and Spring Core Hands-on Exercises (Mandatory)

# Exercise 1: Spring Boot Web Project Creation Using Maven

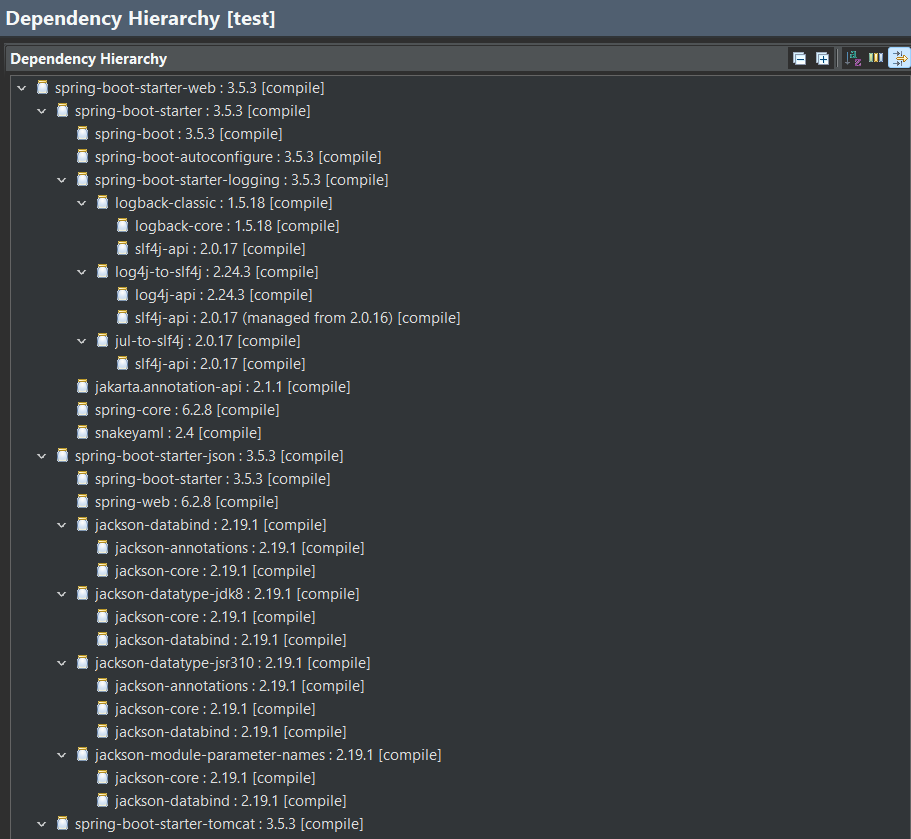
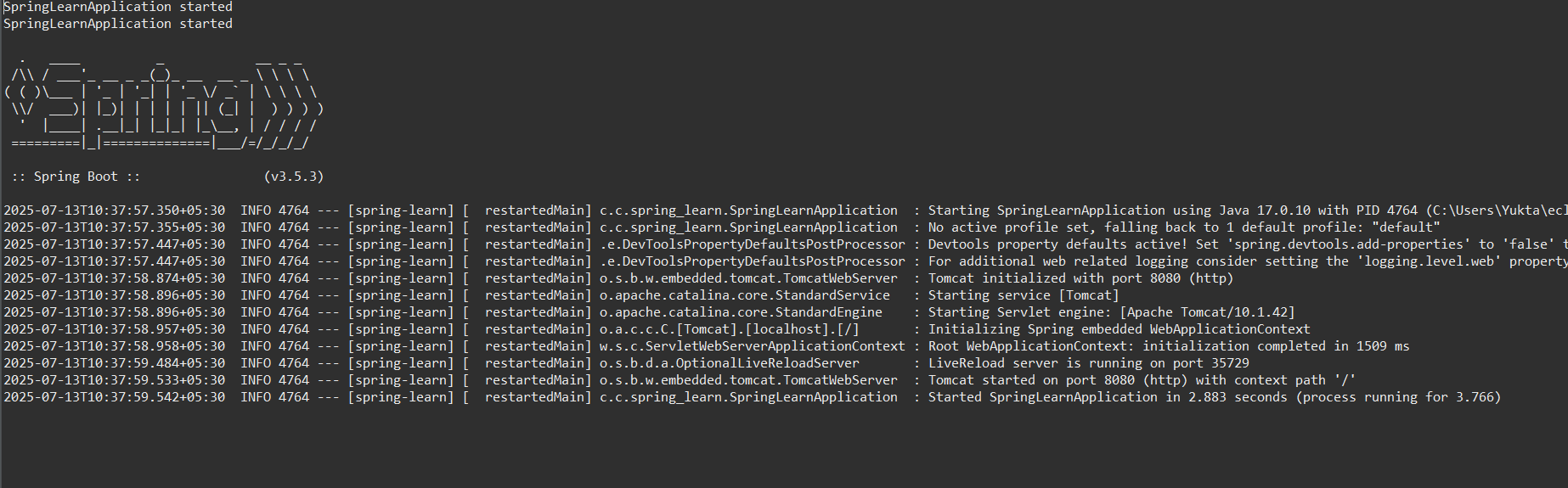
## Objective / Question:

Create a Spring Boot Web Project using Maven with the following:  
- Group: com.cognizant  
- Artifact: spring-learn  
- Dependencies: Spring Web, Spring Boot DevTools  
- Log a message from the main() method and run the application

## Solution / Steps Followed:

1. Visited https://start.spring.io  
2. Entered:  
 - Group: com.cognizant  
 - Artifact: spring-learn  
3. Added Dependencies:  
 - Spring Web  
 - Spring Boot DevTools  
4. Downloaded the generated .zip file  
5. Extracted the folder into eclipse-workspace  
6. Ran the following Maven command in command prompt:  
 mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456  
7. Imported the project into Eclipse using: File > Import > Maven > Existing Maven Project  
8. Added a System.out.println to the main() method to print 'SpringLearnApplicationstarted’

9.verified console output also displayed the Dependency hierarchy



# Exercise 4: Spring Core – Load Country from Spring Configuration XML

## Objective / Question:

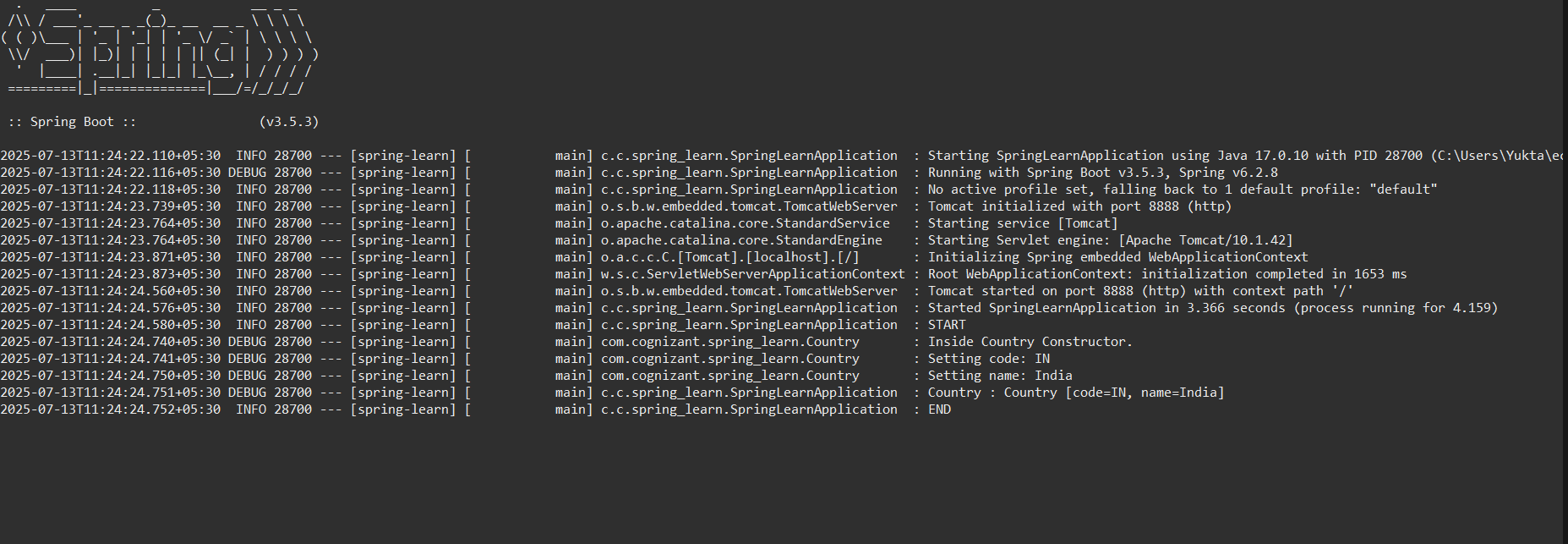
Use Spring’s XML configuration to define a Country bean with ISO code and name.  
Load the bean using ApplicationContext, and log the output from the application with appropriate DEBUG logs.

## Solution / Steps Followed:

1. Created the Country class in src/main/java/com/cognizant/spring\_leearn/Country.java with fields code and name, constructor with debug log, setters/getters with debug log, and toString().  
2. Created country.xml in src/main/resources with the following content:

<bean id="country" class="com.cognizant.spring\_leearn.Country">  
 <property name="code" value="IN" />  
 <property name="name" value="India" />  
</bean>

3. In SpringLearnApplication.java, created displayCountry() method, loaded ApplicationContext, retrieved bean, logged its values, and invoked the method from main().  
4. Configured logging in application.properties to show DEBUG logs:  
 logging.level.com.cognizant.spring\_leearn=debug  
5. Ran the application and verified output logs included constructor, setter, getter, and final object output.



2) Spring Boot REST Hands-on(Mandatory)

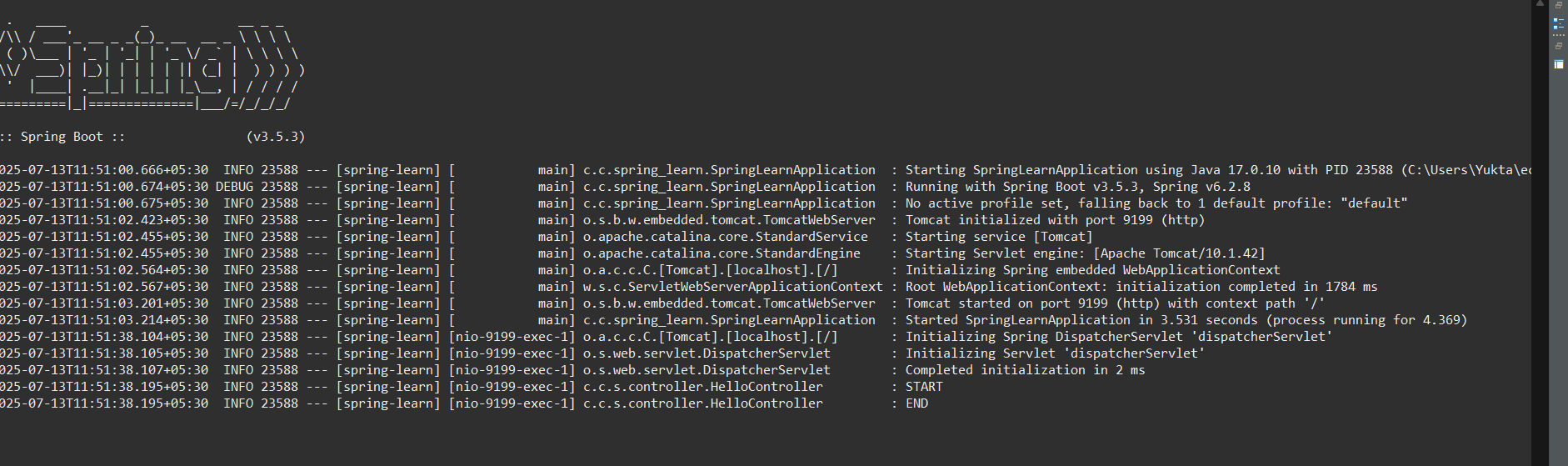
# Exercise 1: Hello World RESTful Web Service

## Objective:

Create a simple REST API that returns "Hello World!!" when called.

## Controller Code:

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");  
 LOGGER.info("END");  
 return "Hello World!!";  
 }  
}



# Exercise 2: Country REST Web Service

## Objective:

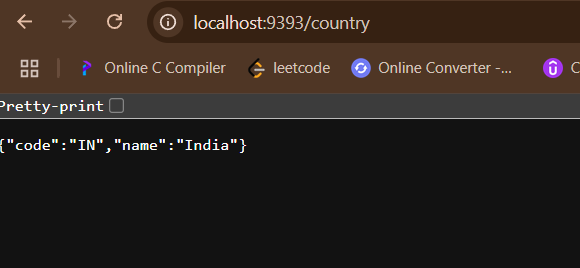
Return a Country object from XML configuration via REST endpoint.

## country.xml:

<bean id="country" class="com.cognizant.spring\_learn.Country">  
 <property name="code" value="IN" />  
 <property name="name" value="India" />  
</bean>

## Controller Code:

package com.cognizant.spring\_learn.controller;  
  
import com.cognizant.spring\_learn.Country;  
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 {  
  
 @RequestMapping("/country")  
 public Country getCountryIndia() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return context.getBean("country", Country.class);  
 }  
}



# Exercise 3: Get Country by Code

## Objective:

Use a list of Country beans from XML and return one based on a path variable.

## country.xml (List Version):

<bean id="countryList" class="java.util.ArrayList">  
 <constructor-arg>  
 <list>  
 <bean class="com.cognizant.spring\_learn.Country">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
 </bean>  
 <bean class="com.cognizant.spring\_learn.Country">  
 <property name="code" value="US"/>  
 <property name="name" value="United States"/>  
 </bean>  
 </list>  
 </constructor-arg>  
</bean>

## CountryService.java:

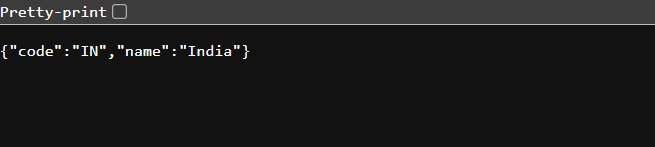
package com.cognizant.spring\_learn.service;  
  
import com.cognizant.spring\_learn.Country;  
import com.cognizant.spring\_learn.service.exception.CountryNotFoundException;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import java.util.List;  
  
public class CountryService {  
  
 public Country getCountry(String code) throws CountryNotFoundException {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 List<Country> countryList = context.getBean("countryList", List.class);  
 return countryList.stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElseThrow(() -> new CountryNotFoundException("Country not found"));  
 }  
}

## CountryNotFoundException.java:

package com.cognizant.spring\_learn.service.exception;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.web.bind.annotation.ResponseStatus;  
  
@ResponseStatus(value = HttpStatus.NOT\_FOUND, reason = "Country not found")  
public class CountryNotFoundException extends Exception {  
 public CountryNotFoundException(String message) {  
 super(message);  
 }  
}

## Controller Method (/countries/{code}):

@GetMapping("/countries/{code}")  
public Country getCountry(@PathVariable String code) throws CountryNotFoundException {  
 CountryService service = new CountryService();  
 return service.getCountry(code);  
}



**Exercise: JWT Authentication in Spring Boot**

In this exercise, we implemented a simple JWT-based authentication system in a Spring Boot application. The goal was to create a login endpoint that returns a JWT token and then use that token to access protected endpoints in the application.

**Task Overview**

1. Create an endpoint `/authenticate` that returns a JWT token.  
2. Allow only authenticated users (with valid JWT) to access `/countries/IN`.  
3. Use Postman or browser + HTML to test the endpoints.

**Implementation Details**

**1. AuthenticationController.java**

This controller contains the `/authenticate` endpoint. If the username and password match ('admin' and 'password'), it generates and returns a JWT token.

***@RestController  
public class AuthenticationController {  
  
 @GetMapping("/authenticate")  
 public String authenticate(@RequestParam String username, @RequestParam String password) {  
 if ("admin".equals(username) && "password".equals(password)) {  
 return JwtUtil.generateToken(username);  
 } else {  
 throw new RuntimeException("Invalid Credentials");  
 }  
 }  
}***

**2. JwtUtil.java**

This utility class generates and validates JWT tokens using the jjwt library.

***public class JwtUtil {  
 private static final String SECRET\_KEY = "secret";  
  
 public static String generateToken(String username) {  
 return Jwts.builder()  
 .setSubject(username)  
 .setIssuedAt(new Date(System.currentTimeMillis()))  
 .setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 30))  
 .signWith(SignatureAlgorithm.HS256, SECRET\_KEY)  
 .compact();  
 }  
  
 public static Claims validateToken(String token) {  
 return Jwts.parser()  
 .setSigningKey(SECRET\_KEY)  
 .parseClaimsJws(token)  
 .getBody();  
 }  
}***

**3. JwtFilter.java**

This filter checks the Authorization header, validates the JWT, and sets the user authentication in the SecurityContext.

***@Component  
public class JwtFilter extends OncePerRequestFilter {  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  
 throws ServletException, IOException {  
  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader != null && authHeader.startsWith("Bearer ")) {  
 String token = authHeader.substring(7);  
 try {  
 Claims claims = JwtUtil.validateToken(token);  
 String username = claims.getSubject();  
  
 UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(username, null, Collections.emptyList());  
 SecurityContextHolder.getContext().setAuthentication(auth);  
 } catch (Exception e) {  
 response.sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Invalid or expired token");  
 return;  
 }  
 }  
  
 filterChain.doFilter(request, response);  
 }  
}***

**4. SecurityConfig.java**

This configuration file disables CSRF, allows unauthenticated access to `/authenticate`, and applies the `JwtFilter` to secure all other endpoints.

***@Configuration  
public class SecurityConfig {  
  
 @Autowired  
 private JwtFilter jwtFilter;  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable())  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/authenticate").permitAll()  
 .anyRequest().authenticated()  
 )  
 .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);  
  
 return http.build();  
 }  
}***

