**Hands on 1**

**Question :**

**Create a Spring Web Project using Maven**   
  
Follow steps below to create a project: 

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.

**Code :**

**SpringLearnApplication.java :**

package com.cognizant.springlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

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

public static void main(String[] args) {

LOGGER.info("START");

SpringApplication.run(SpringLearnApplication.class, args);

LOGGER.info("END");

}

}

**SpringLearnApplicationTests.java :**

package com.cognizant.springlearn;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class SpringLearnApplicationTests {

@Test

void contextLoads() {

}

}

**pom.xml :**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-learn</name>

<description>Demo project for Spring Boot Learn Hands-On</description>

<parent>

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

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

<version>3.2.4</version>

<relativePath/>

</parent>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Boot DevTools -->

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- Spring Web -->

<dependency>

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

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

</dependency>

<!-- Spring Boot Test -->

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Output :**

A black screen with white text

AI-generated content may be incorrect.

**Hands on 4**

**Question:**

Spring Core – Load Country from Spring Configuration XML

An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country.

Code Name

US United States

DE Germany

IN India

JP Japan

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details.

Steps to implement

• Pick any one of your choice country to configure in Spring XML configuration named country.xml.

• Create a bean tag in spring configuration for country and set the property and values

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

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

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

</bean>

• Create Country class with following aspects:

o Instance variables for code and name

o Implement empty parameter constructor with inclusion of debug log within the constructor with log message as “Inside Country Constructor.”

o Generate getters and setters with inclusion of debug with relevant message within each setter and getter method.

o Generate toString() method

• Create a method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country details. ClassPathXmlApplicationContext, ApplicationContext and context.getBean(“beanId”, Country.class). Refer sample code for displayCountry() method below.

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

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

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

• 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.

SME to provide more detailing about the following aspects:

• bean tag, id attribute, class attribute, property tag, name attribute, value attribute

• ApplicationContext, ClassPathXmlApplicationContext

• What exactly happens when context.getBean() is invoked

**Code :**

**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.springlearn.Country">

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

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

</bean>

</beans>

**Country.java:**

package com.cognizant.springlearn;

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("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.springlearn;

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) {

LOGGER.info("START");

displayCountry();

LOGGER.info("END");

}

public static void displayCountry() {

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

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

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

}

}

**Output :**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Hello World RESTful Web Service**

**Hands on:**

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

**Code :**

**HelloController.java**

package com.cognizant.springlearn.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");

String message = "Hello World!!";

LOGGER.info("END");

return message;

}

}

**Output:**

****

**A white background with black dots

AI-generated content may be incorrect.**

**REST - Country Web Service**

**Hands On:**

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

**Code :**

**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.springlearn.Country">

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

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

</bean>

</beans>

**CountryController.java:**

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.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.info("START");

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

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

LOGGER.info("END");

return country;

}

}

**Output :**

**A white background with black and white clouds

AI-generated content may be incorrect.**

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

**Hands On:**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"

}

**Code :**

**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="countryList" class="java.util.ArrayList">

<constructor-arg>

<list>

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

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

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

</bean>

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

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

<property name="name" value="United States"/>

</bean>

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

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

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

</bean>

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

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

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

</bean>

</list>

</constructor-arg>

</bean>

</beans>

**CountryService.java**

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.Country;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

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) 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"));

}

}

**CountryNotFoundException.java:**

package com.cognizant.springlearn.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);

}

}

**CountryController.java:**

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

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

@RestController

public class CountryController {

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

@Autowired

private CountryService countryService;

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

public Country getCountry(@PathVariable String code) throws CountryNotFoundException {

LOGGER.info("START");

Country country = countryService.getCountry(code);

LOGGER.info("END");

return country;

}

}

**Output :**

**A white background with black dots

AI-generated content may be incorrect.**

**Create authentication service that returns JWT**

**Hands On:**

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.

**Code :**

**pom.xml**

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**SecurityConfig.java**

package com.cognizant.springlearn.security;

import org.springframework.context.annotation.Configuration;

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

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

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/authenticate").permitAll()

.anyRequest().authenticated()

.and()

.httpBasic();

}

}

**JwtUtil.java**

package com.cognizant.springlearn.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtUtil {

private String secretKey = "secret"; // Use environment variables in real apps

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

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

.setExpiration(new Date(System.currentTimeMillis() + 60 \* 60 \* 1000)) // 1 hour validity

.signWith(SignatureAlgorithm.HS256, secretKey)

.compact();

}

}

**AuthenticationController.java**

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.util.JwtUtil;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.http.ResponseEntity;

import org.springframework.util.Base64Utils;

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

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

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

import java.nio.charset.StandardCharsets;

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthenticationController {

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

@Autowired

private JwtUtil jwtUtil;

@GetMapping("/authenticate")

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

LOGGER.info("START");

// Extract base64 credentials

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

byte[] decodedBytes = Base64Utils.decodeFromString(base64Credentials);

String decodedCredentials = new String(decodedBytes, StandardCharsets.UTF\_8);

// Split into username and password

String[] userDetails = decodedCredentials.split(":", 2);

String username = userDetails[0];

String password = userDetails[1];

// For demo, hardcoded check. In real apps, check DB or LDAP

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

String token = jwtUtil.generateToken(username);

Map<String, String> response = new HashMap<>();

response.put("token", token);

LOGGER.info("END");

return ResponseEntity.ok(response);

} else {

LOGGER.info("END");

return ResponseEntity.status(401).build();

}

}

}

**Output:**

A white background with black dots

AI-generated content may be incorrect.