**Hands on 1**

# Create a Spring Web Project using Maven

1. SpringLearnApplication.java - Walkthrough the main() method.
2. Purpose of @SpringBootApplication annotation
3. pom.xml
    1. Walkthrough all the configuration defined in XML file
    2. Open 'Dependency Hierarchy' and show the dependency tree.

**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) {

		System.out.println(" We are in SpringLearnApplication main");

		SpringApplication.run(SpringLearnApplication.class, args);

	}

}
```

**Purpose of @SpringBootApplication annotation**

= The @SpringBootApplication annotation is a shortcut that combines three key annotations commonly used in Spring Boot applications: @Configuraton, @EnableAutoConfiguration and @ComponentScan.

@Configuration states that the class contains beans config, the second one states that Spring Boot will auto-configure the application based on the dependencies present in the classpath, and the 3rd one Enables component scanning.

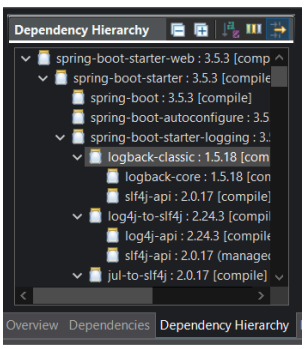
**pom.xml (Walkthrough all the configuration defined in XML file)**

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

<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>
<parent>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-parent</artifactId>
	<version>3.5.3</version>
	<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.cognizant</groupId>
<artifactId>spring-learn</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>spring-learn</name>
<description>Demo project for Spring Boot</description>
<url/>
<licenses>
	<license/>
</licenses>
<developers>
	<developer/>
</developers>
<scm>
	<connection/>
	<developerConnection/>
	<tag/>
	<url/>
</scm>
<properties>
	<java.version>17</java.version>
</properties>
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <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>
```

**Open 'Dependency Hierarchy' and show the dependency tree.**



**Hands on**

**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.**
**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**

**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"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       https://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/util
       https://www.springframework.org/schema/util/spring-util.xsd">
  <bean id="countryUS" class="com.cognizant.spring_learn.Country">
    <property name="code" value="US" />
```

```
    <property name="name" value="United States" />
  </bean>
  <bean id="countryDE" class="com.cognizant.spring_learn.Country">
    <property name="code" value="DE" />
    <property name="name" value="Germany" />
  </bean>
  <bean id="countryIN" class="com.cognizant.spring_learn.Country">
    <property name="code" value="IN" />
    <property name="name" value="India" />
  </bean>
  <bean id="countryJP" class="com.cognizant.spring_learn.Country">
    <property name="code" value="JP" />
    <property name="name" value="Japan" />
  </bean>
  <util:list id="countryList" list-class="java.util.ArrayList">
    <ref bean="countryUS"/>
    <ref bean="countryDE"/>
    <ref bean="countryIN"/>
    <ref bean="countryJP"/>
  </util:list>
</beans>
```

**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("Country Constructor run");
    }
    public String getCode() {
        LOGGER.debug("getCode() is running");
        return code;
    }
    public void setCode(String code) {
        LOGGER.debug("setCode() is runing");
        this.code = code;
    }
    public String getName() {
        LOGGER.debug("getName() is running");
        return name;
    }
    public void setName(String name) {
        LOGGER.debug("setName() is running");
        this.name = name;
    }
    @Override
    public String toString() {
        return "Country [code=" + code + ", name=" + name + "]";
    }
}
```

**SpringLearnApplication.java**

```
package com.cognizant.spring_learn;
import java.util.List;
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.debug("START");
        displayAllCountries();
        LOGGER.debug("END");
    }
    public static void displayAllCountries() {
        ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
        @SuppressWarnings("unchecked")
        List<Country> countryList = (List<Country>) context.getBean("countryList");
        for (Country country : countryList) {
                        System.out.println("Country: " + country);
        }
    }
}
```

**HANDS ON 3 FROM 2-spring-rest-handson**

**Hello World RESTful Web Service**

**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.**

ANS:

**HelloController.java**

```
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.debug("Start sayHello() method");
      String message = "Hello World!!";
      LOGGER.debug("End sayHello() method");
      return message;
   }
}
```

**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);
   }
}
```

**application.properties**

```
spring.application.name=spring-learn
server.port=8083
logging.level.root=DEBUG
```

**HANDS ON QUESTION**

**REST - Country Web Service 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"**

**CountryController.java**

```
package com.cognizant.spring_learn.controller;
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() method hee");
      ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
      Country country = (Country) context.getBean("countryIN");
      LOGGER.debug("End getCountryIndia() method here");
      return country;
   }
}
```

**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);

   }

}
```

**application.properties**

```
spring.application.name=spring-learn

server.port=8083

logging.level.root=D
```

**HANDS ON QUESTION**

**REST - Get country based on country code 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"

}
```

**CountryService.java:**

```
package com.cognizant.spring_learn.service;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.stereotype.Service;
import com.cognizant.spring_learn.Country;
@Service
public class CountryService {
   public Country getCountry(String code) {
      ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");
      List<Country> countryList = (List<Country>) context.getBean("countryList");
      Optional<Country> matchedCountry = countryList.stream()
            .filter(c -> c.getCode().equalsIgnoreCase(code))
            .findFirst();0
      if (matchedCountry.isPresent()) {
         return matchedCountry.get();
      } else {
         throw new RuntimeException("Country with code " + code + " not found");
      }
   }
}
```

**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);
    }
}
```

**application.properties**

```
spring.application.name=spring-learn
server.port=8083
logging.level.root=DEBUG
```

**CountryController.java**

```
package com.cognizant.spring_learn.controller;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.cognizant.spring_learn.Country;
import com.cognizant.spring_learn.service.CountryService;
@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) {
        LOGGER.debug("Start getCountry()");
        Country country = countryService.getCountry(code);
        LOGGER.debug("End getCountry()");
        return country;
    }
}
```

**HANDS ON 5**

**Create authentication service that returns JWT 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.**

- **Create authentication controller and configure it in SecurityConfig**

**AuthenticationController.java**

```
package com.cognizant.spring_learn.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthenticationController {

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

   @GetMapping("/authenticate")

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

      LOGGER.info("Start authenticate() method from here");

      LOGGER.debug("Authorization Header: {}", authHeader);

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

      tokenMap.put("token", "");

      LOGGER.info("End authenticate() here");
```

```
        return tokenMap;
    }
}
```

**SecurityConfig.java**

```
package com.cognizant.spring_learn.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.config.Customizer;
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf(csrf -> csrf.disable()).authorizeHttpRequests(auth -> auth
                .requestMatchers("/countries").hasRole("USER")
                .requestMatchers("/authenticate").hasAnyRole("USER", "ADMIN")
                .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults());
        return http.build();
    }
}
```

**Testing curl command**

curl -s -u user:pwd http://localhost:8090/authenticate

**output:**

```
>curl -u user:pwd http://localhost:8090/authenticate
{"token":""}
```

Check if Authorization header value is displayed with "Basic" prefix and Base64 encoding of "user:pwd" = yes

- **Read Authorization header and decode the username and password**

The AuthenticationController class is updated with a new private method.

```
package com.cognizant.spring_learn.controller;
import org.springframework.web.bind.annotation.*;
import java.util.Base64;
import java.util.HashMap;
import java.util.Map;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
@RestController
public class AuthenticationController {
   private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);
   @GetMapping("/authenticate")
   public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {
      LOGGER.info("authenticate() called");
      LOGGER.debug("Authorization Header is received: " + authHeader);
      String username = extractUsername(authHeader);
      LOGGER.debug("Username extracted: " + username);
      Map<String, String> response = new HashMap<>();
      response.put("token", "");
      LOGGER.info("authenticate() finished yee");
      return response;
   }
   private String extractUsername(String authHeader) {
  LOGGER.debug("Inside extractUsername() meth0od");
      try {
         // Remove "Basic " from the beginning
         String encoded = authHeader.substring(6);
         LOGGER.debug("Encoded credentials: " + encoded);
         // Decode the Base64 encoded string
```

```
        byte[] decodedBytes = Base64.getDecoder().decode(encoded);
        String decoded = new String(decodedBytes);
        LOGGER.debug("Decoded string: " + decoded);
        // Split at ':' to separate username and password
        String[] parts = decoded.split(":");
        return parts[0];
      } catch (Exception e) {
        LOGGER.error("decodinf credentials caused error ", e);
        return null;
      }
   }
}
```

**Test:**

curl -u user:pwd http://localhost:8090/authenticate

Output:

```
>curl -u user:pwd http://localhost:8090/authenticate
{"token":""}
>
```

**NOW CHECKING THE LOGS:**

```
icationController          : authenticate() called
icationController          : Authorization Header is received: Basic dXNlcjpwd2Q=
icationController          : Inside extractUsername() meth0od
icationController          : Encoded credentials: dXNlcjpwd2Q=
icationController          : Decoded string: user:pwd
icationController          : Username extracted: user
icationController          : authenticate() finished yee
sponseBodyMethodProcessor : Using 'application/json', given [*/*] and supported [application/json, app
sponseBodyMethodProcessor : Writing [{token=}]
.DispatcherServlet         : Completed 200 OK
.http11.Http11Processor    : Error parsing HTTP request header
```

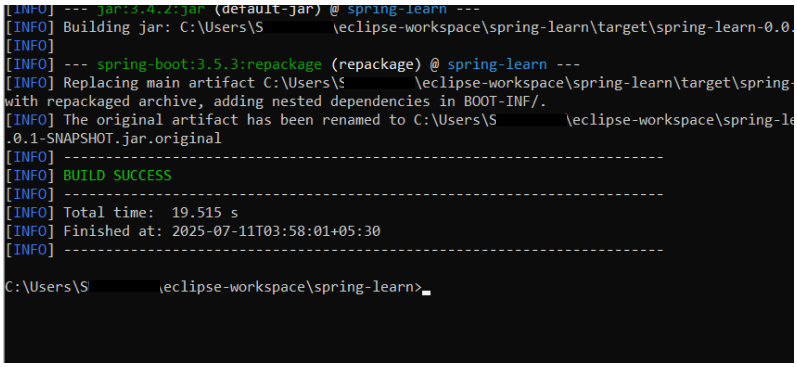
- **Generate token based on the user**

Include JWT library by including the following maven dependency.

```
<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt</artifactId>
<version>0.9.0</version>
</dependency>
```

· **After inclusion in pom.xml, run the maven package command line and update the project in Eclipse. View the dependency tree and check if the library is added.**

**mvn clean install -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttp.proxyUser=123456**



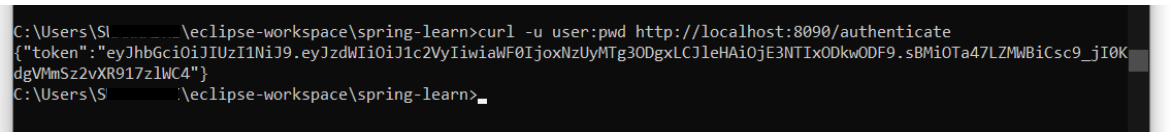

**UPDATED AuthenticationController.java**

```
package com.cognizant.spring_learn.controller;

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

import java.util.Base64;

import java.util.HashMap;

import java.util.Map;

import java.util.Date;

import io.jsonwebtoken.JwtBuilder;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@RestController

public class AuthenticationController {

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

   @GetMapping("/authenticate")

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

      LOGGER.info("authenticate() called");

      LOGGER.debug("Authorization Header is received: " + authHeader);

      String username = extractUsername(authHeader);

      LOGGER.debug("Username extracted: " + username);
```

```
        String token = generateJwt(username);
        LOGGER.debug("Generated token: " + token);
        Map<String, String> response = new HashMap<>();
        response.put("token", token);
        LOGGER.info("authenticate() finished yee");
        return response;
    }
    private String extractUsername(String authHeader) {
        LOGGER.debug("Inside extractUsername() meth0od");
        try {
            String encoded = authHeader.substring(6); // remove "Basic "
            LOGGER.debug("Encoded credentials: " + encoded);
            byte[] decodedBytes = Base64.getDecoder().decode(encoded);
            String decoded = new String(decodedBytes);
            LOGGER.debug("Decoded string: " + decoded); // user:pwd
            String[] parts = decoded.split(":");
            return parts[0];
        } catch (Exception e) {
            LOGGER.error("decodinf credentials caused error ", e);
            return null;
        }
    }
    private String generateJwt(String user) {
        JwtBuilder builder = Jwts.builder();
        builder.setSubject(user);
        builder.setIssuedAt(new Date());
        builder.setExpiration(new Date((new Date()).getTime() + 1200000)); // 20 mins time
        builder.signWith(SignatureAlgorithm.HS256, "secretkey");
        return builder.compact();
    }
}
```

**NEXT WE RUN THE MAIN METHOD AND TEST,  THIS IS THE OUTPUT:**

curl -u user:pwd http://localhost:8090/authenticate




**GENERATED TOKEN IS RETURNED.**