

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.

Answer:

SpringLearnApplication.java:

package com.cognizant.springlearn;

Output:



2. src/main/resources

"This folder contains configuration files such as application.properties. We can add DB configs, logging, and other settings here."

3. src/test/java

"This folder is for writing unit and integration test cases using frameworks like JUnit or Mockito."

4. SpringLearnApplication.java (main class)

"This is the entry point of the Spring Boot app. Inside main(), we use SpringApplication.run() to bootstrap the app. I've added logging to confirm app startup and shutdown."

5. Purpose of @SpringBootApplication

"This annotation combines:

- @Configuration (for defining configuration)
- @EnableAutoConfiguration (for auto-configuring beans)
- @ComponentScan (for scanning components)

It enables automatic setup of the Spring app with minimal code."

6. pom.xml Walkthrough (in Eclipse)

In Eclipse, go to pom.xml → Select **pom.xml** tab:

- Shows project's Maven coordinates: groupId, artifactId, version.
 - Contains dependencies like:
 - spring-boot-starter-web → For REST APIs
 - spring-boot-devtools → For auto-reloading
 - spring-boot-starter-test → For testing
-

Dependency Hierarchy Tab (in Eclipse)

When you open the **Dependency Hierarchy** tab in Eclipse:

Here we can see all direct and transitive dependencies clearly.

For example, spring-boot-starter-web internally pulls spring-web, spring-boot-starter, spring-boot-autoconfigure, and tomcat.

It helps identify dependency conflicts or versions.

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:
 - 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
- 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

Answer:

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>
```

Country.java

```
package com.cognizant.springlearn;
```

```
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
```

```
public class Country {
```

```
    private String code;
```

```
private String name;  
private static final Logger LOGGER = LoggerFactory.getLogger(Country.class);
```

```
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.support.ClassPathXmlApplicationContext;

public class SpringLearnApplication {

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

    public static void main(String[] args) {

        LOGGER.info("START");

        try (ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("country.xml")) {

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

            System.out.println("Country: " + country);

        }

        LOGGER.info("END");

    }

}

```

Spring XML Configuration Tags :

- <bean> tag: Declares a Spring bean (Java object) that is managed by the Spring container.
- id attribute: Provides a unique identifier to reference the bean in the configuration.
- class attribute: Specifies the fully qualified class name of the Java bean to be instantiated.
- <property> tag: Sets a value for a property of the bean using its setter method.
- name attribute: Indicates the name of the bean property to be set (matches the setter method without "set").
- value attribute: Defines the actual value that will be assigned to the property.

Spring Context Details :

- **ApplicationContext:** It is the Spring container that holds, manages, and provides access to beans.
- **ClassPathXmlApplicationContext:** Loads the Spring XML configuration file from the classpath to initialize the container.
- **context.getBean("id"):** Retrieves the bean object with the specified id from the Spring container.

When context.getBean() is Called:

1. Spring locates the bean by id.
2. Spring creates the object (if not already created).
3. Spring injects properties (calls setter methods with values from XML).
4. Returns the bean object to you.

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.

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

Answer:

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.info("START sayHello()");
```

```
        String message = "Hello World!!";
```

```
        LOGGER.info("END sayHello()");
```

```
        return message;
```

```
    }
```

```
}
```

Application Properties

```
server.port=8083
```

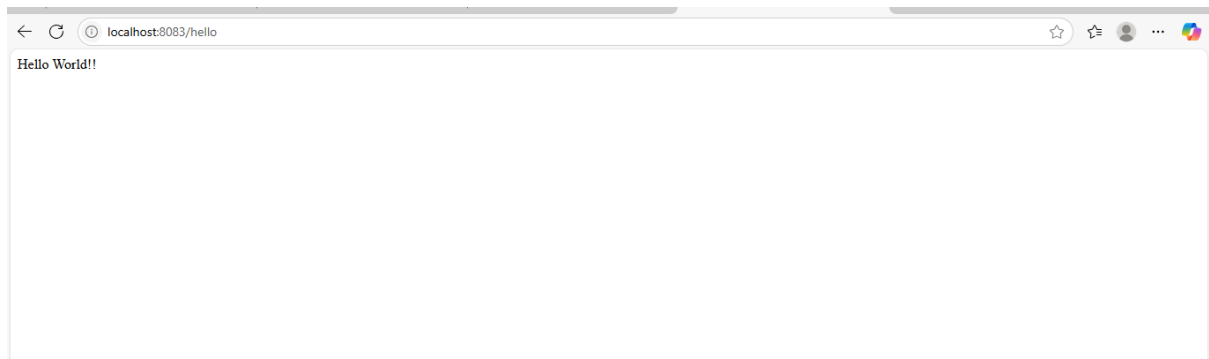
Output:



```

:: Spring Boot ::                (v3.5.3)

2025-07-07T21:33:54.895+05:30 INFO 114732 --- [ restartedMain] c.c.springlearn.SpringLearnApplication : Starting SpringLearnApplication using Java 17.0.10 with PID 114732 (C:\Users\va
2025-07-07T21:33:54.898+05:30 DEBUG 114732 --- [ restartedMain] c.c.springlearn.SpringLearnApplication : Running with Spring Boot v3.5.3, Spring v6.2.8
2025-07-07T21:33:54.899+05:30 INFO 114732 --- [ restartedMain] c.c.springlearn.SpringLearnApplication : No active profile set, falling back to 1 default profile: "default"
2025-07-07T21:33:55.182+05:30 INFO 114732 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'fr
2025-07-07T21:33:55.182+05:30 INFO 114732 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' pr
2025-07-07T21:33:56.611+05:30 INFO 114732 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8083 (http)
2025-07-07T21:33:56.635+05:30 INFO 114732 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-07-07T21:33:56.698+05:30 INFO 114732 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.42]
2025-07-07T21:33:56.781+05:30 INFO 114732 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-07-07T21:33:57.250+05:30 INFO 114732 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: Initialization completed in 1594 ms
2025-07-07T21:33:57.318+05:30 INFO 114732 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : LiveReload server is running on port 35729
2025-07-07T21:33:57.336+05:30 INFO 114732 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8083 (http) with context path '/'
2025-07-07T21:34:10.586+05:30 INFO 114732 --- [nio-8083-exec-1] c.c.springlearn.SpringLearnApplication : Started SpringLearnApplication in 3.417 seconds (process running for 4.024)
2025-07-07T21:34:10.586+05:30 INFO 114732 --- [nio-8083-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-07-07T21:34:10.590+05:30 INFO 114732 --- [nio-8083-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-07-07T21:34:10.785+05:30 INFO 114732 --- [nio-8083-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 4 ms
2025-07-07T21:34:10.785+05:30 INFO 114732 --- [nio-8083-exec-1] c.c.s.controller.HelloController : START sayHello()
2025-07-07T21:34:10.785+05:30 INFO 114732 --- [nio-8083-exec-1] c.c.s.controller.HelloController : END sayHello()
  
```



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"
}
```

SME to explain the following aspects:

- What happens in the controller method?
- How the bean is converted into JSON response?
- 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

Answer:

SpringLearnApplication.java:

package com.cognizant.springlearn;

```
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);
    }
}
```

CountryController.java:

```
package com.cognizant.springlearn.controller;
```

```
import com.cognizant.spring.learn.model.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");
```

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

```
        return country;
```

```
    }
```

```
}
```

Country.java:

```
package com.cognizant.spring.learn.model;
```

```

public class Country {

    private String code;

    private String 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;

    }

}

```

Country.xml:

```

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

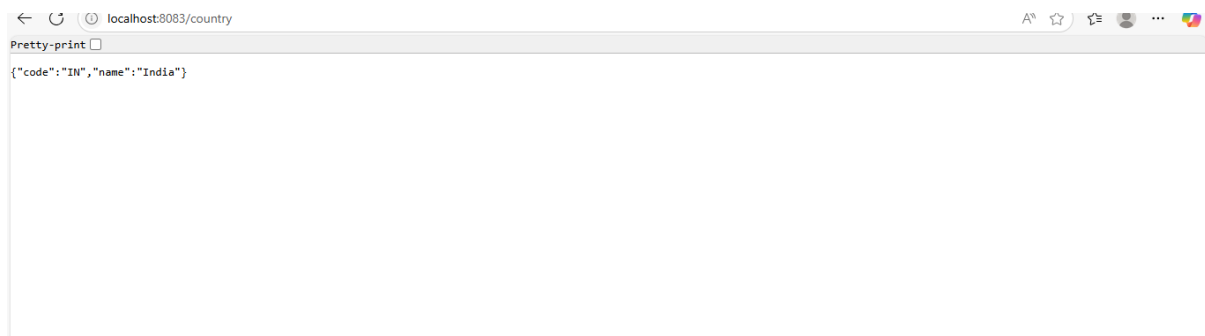
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

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

</beans>

```

Output:



• What happens in the controller method?

In the controller method, when the URL /country is accessed, the method `getCountryIndia()` gets called.

Inside this method, the Spring container loads the country bean from the XML configuration file (country.xml) using `ClassPathXmlApplicationContext`.

This bean contains details of India such as country code and name.

The method returns the Country object, which contains these values.

• How the bean is converted into JSON response?

In the Spring Boot application, the returned Country object is automatically converted into a JSON response by Spring's built-in HTTP message converter.

Spring Boot uses the **Jackson** library internally (which comes with spring-boot-starter-web) to perform this conversion.

There is no need for manual JSON conversion; it happens automatically based on the object returned by the method.

• In network tab of developer tools show the HTTP header details received:

1. Open your browser (such as Chrome) and press F12 to open Developer Tools.
2. Go to the **Network** tab and access the URL: `http://localhost:8083/country`.
3. Click on the /country request shown in the network list.
4. Under the **Headers** section, in **Response Headers**, you will see:

Content-Type: application/json

• In Postman click on "Headers" tab to view the HTTP header details received:

1. Open Postman and send a **GET** request to `http://localhost:8083/country`.
2. Once the response is received, click on the **Headers** tab under the response section.

3. You will find HTTP response headers like:

Content-Type: application/json

Content-Length: <depends on response>

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 Implementation: 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"  
}
```

Answer:

Country.java:

package com.cognizant.spring.learn.model;

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;  
}  
}
```

SpringLearnApplication.java:

```
package com.cognizant.springlearn;
```

```
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);  
}  
}
```

CountryController.java:

```
package com.cognizant.springlearn.controller;  
  
import com.cognizant.spring.learn.model.Country;  
import com.cognizant.springlearn.service.CountryService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.*;
```

@RestController

```
public class CountryController {  
  
    @Autowired  
    private CountryService countryService;  
  
    @GetMapping("/countries/{code}")  
    public Country getCountry(@PathVariable String code) {  
        return countryService.getCountry(code);  
    }  
}
```

CountryService.java:

```
package com.cognizant.springlearn.service;  
  
import com.cognizant.spring.learn.model.Country;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.stereotype.Service;  
  
import java.util.List;
```


@Service

```
public class CountryService {  
  
    public Country getCountry(String code) {  
  
        try (ClassPathXmlApplicationContext context = new  
ClassPathXmlApplicationContext("country.xml")) {  
  
            List<Country> countries = (List<Country>) context.getBean("countryList");  
  
            return countries.stream()  
  
                .filter(country -> country.getCode().equalsIgnoreCase(code))  
  
                .findFirst()  
  
                .orElse(null);  
  
        }  
    }  
}
```

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.spring.learn.model.Country">  
                    <property name="code" value="IN" />  
                    <property name="name" value="India" />  
                </bean>  
                <bean class="com.cognizant.spring.learn.model.Country">  
                    <property name="code" value="US" />  
                    <property name="name" value="United States" />  
                </bean>  
            </list>  
        </constructor-arg>  
    </bean>  
</beans>
```

```
        </bean>

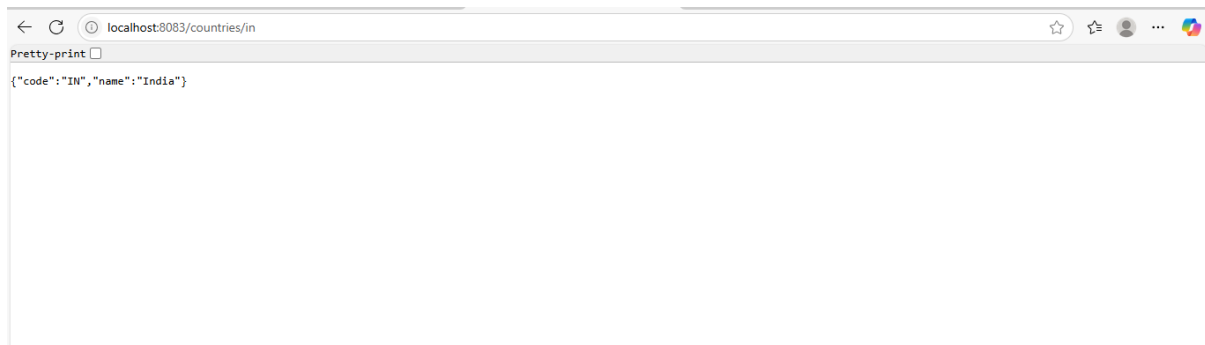
    </list>

    </constructor-arg>

</bean>

</beans>
```

Output:



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.eyJzdWl0OiJ1c2VyliwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOiJlNzAzODAzR9.2NzR9.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.

Answer:

AuthenticationApplication.java

```
package com.example.authservice;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
```

```
public class AuthserviceApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(AuthserviceApplication.class, args);
```

```
    }
```

```
}
```

JwtRequestFilter.java:

```
package com.example.authservice.config;
```

```
import com.example.authservice.util.JwtUtil;
```

```
import io.jsonwebtoken.Claims;
```

```
import io.jsonwebtoken.JwtException;
```

```
import jakarta.servlet.FilterChain;
```

```
import jakarta.servlet.ServletException;
```

```
import jakarta.servlet.http.HttpServletRequest;
```

```
import jakarta.servlet.http.HttpServletResponse;
```

```
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
```

```
import org.springframework.security.core.context.SecurityContextHolder;
```

```
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
```

```
import org.springframework.web.filter.OncePerRequestFilter;
```

```
import java.io.IOException;
```

```
public class JwtRequestFilter extends OncePerRequestFilter {
```

@Override

```
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain)
    throws ServletException, IOException {

    final String authorizationHeader = request.getHeader("Authorization");

    if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
        String jwt = authorizationHeader.substring(7);
        try {
            Claims claims = JwtUtil.parseToken(jwt);
            String username = claims.getSubject();

            UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(username, null, null);
            authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authToken);

        } catch (JwtException e) {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            return;
        }
    }

    filterChain.doFilter(request, response);
}
```

SecurityConfig.java:

```
package com.example.authservice.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.web.authentication.UsernamePasswordAuthenticationFilter;
```

@Configuration

```
public class SecurityConfig {
```

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/authenticate").permitAll()
            .anyRequest().authenticated()
        )
        .addFilterBefore(new JwtRequestFilter(), UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
}
```

AuthenticationController.java:

```
package com.example.authservice.controller;
```

```
import com.example.authservice.util.JwtUtil;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```
import java.util.Base64;
```

@RestController

```
public class AuthenticationController {
```

```

@RequestMapping("/authenticate")

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

    if (authHeader != null && authHeader.startsWith("Basic")) {

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

        byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);

        String credentials = new String(credDecoded);

        String[] values = credentials.split(":", 2);

        String username = values[0];

        String password = values[1];

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

            String token = JwtUtil.generateToken(username);

            return ResponseEntity.ok().body("{\"token\":\"" + token + "\"}");

        } else {

            return ResponseEntity.status(401).body("Invalid credentials");

        }

    } else {

        return ResponseEntity.badRequest().body("Missing Authorization header");

    }

}

```

HelloController.java:

```

package com.example.authservice.controller;

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

@RestController

public class HelloController {

```

```

    @GetMapping("/hello")
    public String hello() {
        return "Hello, JWT Secured World!";
    }
}

```

JwtUtil.java:

```

package com.example.authservice.util;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;

import java.util.Date;

public class JwtUtil {

    private static final String SECRET_KEY = "mysecretkeymysecretkeymysecretkey";

    public static String generateToken(String username) {
        long currentTimeMillis = System.currentTimeMillis();
        return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new Date(currentTimeMillis))
            .setExpiration(new Date(currentTimeMillis + 1000 * 60 * 60)) // 1 hour expiry
            .signWith(Keys.hmacShaKeyFor(SECRET_KEY.getBytes()), SignatureAlgorithm.HS256)
            .compact();
    }

    public static Claims parseToken(String token) {

```

```

return Jwts.parserBuilder()

    .setSigningKey(SECRET_KEY.getBytes())

    .build()

    .parseClaimsJws(token)

    .getBody();
}
}

```

Output:

```

AxOJsDPJSWOLi-0oQKDxmLH9mPN
Hello, JWT Secured World!

```

Problem Statement - Display Employee List and Edit Employee form using RESTful Web Service

In the previous angular module, we developed a screen that lists employees and it was populated with hard coded values. Now this angular application has be changed to get the data from RESTful Web Service developed in Spring. The following are the high level activities that needs to be done to accomplish this:

- Create static employee list data using spring xml configuration
- Create a REST Service that reads data from xml configuration and returns it
- Make changes in angular component to consume the created REST Service

Once above activities are completed, clicking on the Edit button against each employee should display Edit Employee form with values retrieved from RESTful Web Service. This will also involve activities similar to the one specified above.

NOTE: There is no specific activity as part of this hands on, refer the next hands ons that covers above three activities in detail.

Answer:

AuthserviceApplication.java:

```

package com.example.authservice;

```

```

import org.springframework.boot.SpringApplication;

```

```

import org.springframework.boot.autoconfigure.SpringBootApplication;

```



```

        throws ServletException, IOException {

    final String authorizationHeader = request.getHeader("Authorization");

    if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
        String jwt = authorizationHeader.substring(7);
        try {
            Claims claims = JwtUtil.parseToken(jwt);
            String username = claims.getSubject();

            UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(username, null, null);
            authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authToken);

        } catch (JwtException e) {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            return;
        }
    }

    filterChain.doFilter(request, response);
}
}

```

SecurityConfig.java:

```

package com.example.authservice.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.web.authentication.UsernamePasswordAuthenticationFilter;
```

```
@Configuration
```

```
public class SecurityConfig {
```

```
    @Bean
```

```
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
```

```
        http.csrf(csrf -> csrf.disable())
```

```
        .authorizeHttpRequests(auth -> auth
```

```
            .requestMatchers("/authenticate").permitAll()
```

```
            .anyRequest().authenticated()
```

```
        )
```

```
        .addFilterBefore(new JwtRequestFilter(), UsernamePasswordAuthenticationFilter.class);
```

```
        return http.build();
```

```
    }
```

```
}
```

```
AuthenticationController.java:
```

```
package com.example.authservice.controller;
```

```
import com.example.authservice.util.JwtUtil;
```

```
import org.springframework.http.ResponseEntity;
```

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

```
import java.util.Base64;
```

```
@RestController
```

```
public class AuthenticationController {
```

```
    @RequestMapping("/authenticate")
```

```
    public ResponseEntity<?> authenticate(@RequestHeader("Authorization") String authHeader) {
```

```

if (authHeader != null && authHeader.startsWith("Basic")) {
    String base64Credentials = authHeader.substring("Basic".length()).trim();
    byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);
    String credentials = new String(credDecoded);
    String[] values = credentials.split(":", 2);
    String username = values[0];
    String password = values[1];

    if ("user".equals(username) && "pwd".equals(password)) {
        String token = JwtUtil.generateToken(username);
        return ResponseEntity.ok().body("{\"token\":\"" + token + "\"}");
    } else {
        return ResponseEntity.status(401).body("Invalid credentials");
    }
} else {
    return ResponseEntity.badRequest().body("Missing Authorization header");
}
}

```

HelloController.java:

```

package com.example.authservice.controller;

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

@RestController

public class HelloController {

    @GetMapping("/hello")
    public String hello() {

```

```
        return "Hello, JWT Secured World!";
    }
}
```

JwtUtil.java:

```
package com.example.authservice.util;
```

```
import io.jsonwebtoken.Claims;
```

```
import io.jsonwebtoken.Jwts;
```

```
import io.jsonwebtoken.SignatureAlgorithm;
```

```
import io.jsonwebtoken.security.Keys;
```

```
import java.util.Date;
```

```
public class JwtUtil {
```

```
    private static final String SECRET_KEY = "mysecretkeymysecretkeymysecretkey";
```

```
    public static String generateToken(String username) {
```

```
        long currentTimeMillis = System.currentTimeMillis();
```

```
        return Jwts.builder()
```

```
            .setSubject(username)
```

```
            .setIssuedAt(new Date(currentTimeMillis))
```

```
            .setExpiration(new Date(currentTimeMillis + 1000 * 60 * 60))
```

```
            .signWith(Keys.hmacShaKeyFor(SECRET_KEY.getBytes()), SignatureAlgorithm.HS256)
```

```
            .compact();
```

```
    }
```

```
    public static Claims parseToken(String token) {
```

```
        return Jwts.parserBuilder()
```

```
            .setSigningKey(SECRET_KEY.getBytes())
```

```
            .build()
```

```

        .parseClaimsJws(token)
        .getBody();
    }
}

edit-employee.component.html
<h2>Edit Employee</h2>
<form *ngIf="employee">
    <label>ID:</label> {{ employee.id }} <br>
    <label>Name:</label>
    <input type="text" [(ngModel)]="employee.name" name="name"> <br>
    <label>Salary:</label>
    <input type="number" [(ngModel)]="employee.salary" name="salary"> <br>
    <label>Permanent:</label>
    <input type="checkbox" [(ngModel)]="employee.permanent" name="permanent"> <br>
    <button>Save (Demo Only)</button>
</form>

edit-employee.component.ts
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { ActivatedRoute } from '@angular/router';

@Component({
    selector: 'app-edit-employee',
    standalone: true,
    imports: [CommonModule, FormsModule],
    templateUrl: './edit-employee.component.html',
    styleUrls: ['./edit-employee.component.css']
})
export class EditEmployeeComponent {
    employee: any = { id: 0, name: '', salary: 0, permanent: false };

```

```
constructor(private route: ActivatedRoute) {}
```

```
ngOnInit(): void {  
  const id = Number(this.route.snapshot.paramMap.get('id'));  
  this.employee = {  
    id: id,  
    name: 'Employee ' + id,  
    salary: 50000,  
    permanent: true  
  };  
}  
}
```

edit-employee.spec.ts

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
```

```
import { EditEmployee } from './edit-employee';
```

```
describe('EditEmployee', () => {  
  let component: EditEmployee;  
  let fixture: ComponentFixture<EditEmployee>;
```

```
  beforeEach(async () => {  
    await TestBed.configureTestingModule({  
      imports: [EditEmployee]  
    })  
    .compileComponents();
```

```
    fixture = TestBed.createComponent(EditEmployee);  
    component = fixture.componentInstance;  
    fixture.detectChanges();
```

```
});
```

```
it('should create', () => {  
  expect(component).toBeTruthy();  
});
```

```
});
```

edit-employee.ts

```
import { Component } from '@angular/core';
```

```
@Component({  
  selector: 'app-edit-employee',  
  imports: [],  
  templateUrl: './edit-employee.html',  
  styleUrls: ['./edit-employee.css']  
})  
export class EditEmployee {  
  
}
```

employee-list.component.html

```
<h2>Employee List</h2>  
<table border="1">  
  <tr>  
    <th>ID</th>  
    <th>Name</th>  
    <th>Salary</th>  
    <th>Permanent</th>  
    <th>Actions</th>  
  </tr>  
  <tr *ngFor="let employee of employees">  
    <td>{{ employee.id }}</td>
```



```

<td>{{ employee.name }}</td>
<td>{{ employee.salary }}</td>
<td>{{ employee.permanent }}</td>
<td><button (click)="editEmployee(employee.id)">Edit</button></td>
</tr>
</table>

```

employee-list.component.ts

```

import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule, Router } from '@angular/router';

```

```

@Component({
  selector: 'app-employee-list',
  standalone: true, // Standalone Component (No need to declare in NgModule)
  imports: [CommonModule, RouterModule],
  templateUrl: './employee-list.component.html',
  styleUrls: ['./employee-list.component.css']
})

```

```

export class EmployeeListComponent {
  employees = [
    { id: 1, name: 'John Doe', salary: 50000, permanent: true },
    { id: 2, name: 'Jane Smith', salary: 60000, permanent: false },
  ];
}

```

```

constructor(private router: Router) {}

```

```

editEmployee(id: number): void {
  this.router.navigate(['/edit', id]);
}

```

```

}

```

employee-list.spec.ts

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
```

```
import { EmployeeList } from './employee-list';
```

```
describe('EmployeeList', () => {
```

```
  let component: EmployeeList;
```

```
  let fixture: ComponentFixture<EmployeeList>;
```

```
  beforeEach(async () => {
```

```
    await TestBed.configureTestingModule({
```

```
      imports: [EmployeeList]
```

```
    })
```

```
    .compileComponents();
```

```
    fixture = TestBed.createComponent(EmployeeList);
```

```
    component = fixture.componentInstance;
```

```
    fixture.detectChanges();
```

```
  });
```

```
  it('should create', () => {
```

```
    expect(component).toBeTruthy();
```

```
  });
```

```
});
```

employee-list.ts

```
import { Component } from '@angular/core';
```

```
@Component({
```

```
  selector: 'app-employee-list',
```

```
  imports: [],
```

```
    templateUrl: './employee-list.html',
    styleUrls: ['./employee-list.css']
  })
  export class EmployeeList {

  }
```

```
app.component.html
<h1>{{ title }}</h1>
<router-outlet></router-outlet>
```

```
app.component.ts
import { Component } from '@angular/core';
import { RouterOutlet, Routes } from '@angular/router';
import { provideRouter } from '@angular/router';
import { importProvidersFrom } from '@angular/core';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { CommonModule } from '@angular/common';
```

```
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [
    CommonModule,
    RouterOutlet
  ],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Employee App';
}
```

```
}
```

app.config.ts

```
import { ApplicationConfig, provideBrowserGlobalErrorListeners, provideZoneChangeDetection }  
from '@angular/core';
```

```
import { provideRouter } from '@angular/router';
```

```
import { routes } from './app.routes';
```

```
export const appConfig: ApplicationConfig = {  
  providers: [  
    provideBrowserGlobalErrorListeners(),  
    provideZoneChangeDetection({ eventCoalescing: true }),  
    provideRouter(routes)  
  ]  
};
```

app.routes.ts

```
import { Routes } from '@angular/router';
```

```
export const routes: Routes = [  
  {  
    path: '',  
    loadComponent: () => import('./employee-list/employee-list.component').then(m =>  
m.EmployeeListComponent)  
  },  
  {  
    path: 'edit/:id',  
    loadComponent: () => import('./edit-employee/edit-employee.component').then(m =>  
m.EditEmployeeComponent)  
  }  
];
```

app.spec.ts

```
import { TestBed } from '@angular/core/testing';
```

```
import { App } from './app';
```

```
describe('App', () => {  
  beforeEach(async () => {  
    await TestBed.configureTestingModule({  
      imports: [App],  
    }).compileComponents();  
  });
```

```
  it('should create the app', () => {  
    const fixture = TestBed.createComponent(App);  
    const app = fixture.componentInstance;  
    expect(app).toBeTruthy();  
  });
```

```
  it('should render title', () => {  
    const fixture = TestBed.createComponent(App);  
    fixture.detectChanges();  
    const compiled = fixture.nativeElement as HTMLElement;  
    expect(compiled.querySelector('h1')?.textContent).toContain('Hello, employee-app');  
  });  
});
```

app.ts

```
import { Component } from '@angular/core';  
import { RouterOutlet } from '@angular/router';
```

```
@Component({  
  selector: 'app-root',  
  imports: [RouterOutlet],  
  templateUrl: './app.html',  
  styleUrls: ['./app.css']  
})
```

```

    })

    export class App {
        protected title = 'employee-app';
    }

    employee.service.ts

    import { Injectable } from '@angular/core';
    import { HttpClient } from '@angular/common/http';
    import { Observable } from 'rxjs';

    export interface Employee {
        id: number;
        name: string;
        salary: number;
        permanent: boolean;
    }

    @Injectable({
        providedIn: 'root'
    })

    export class EmployeeService {
        private baseUrl = 'http://localhost:8080/employees';

        constructor(private http: HttpClient) { }

        getEmployees(): Observable<Employee[]> {
            return this.http.get<Employee[]>(this.baseUrl);
        }

        getEmployeeById(id: number): Observable<Employee> {
            return this.http.get<Employee>(`${this.baseUrl}/${id}`);
        }
    }

```

```

}

employee.spec.ts

import { TestBed } from '@angular/core/testing';

import { Employee } from './employee';

describe('Employee', () => {
  let service: Employee;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(Employee);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});

```

```

employee.ts

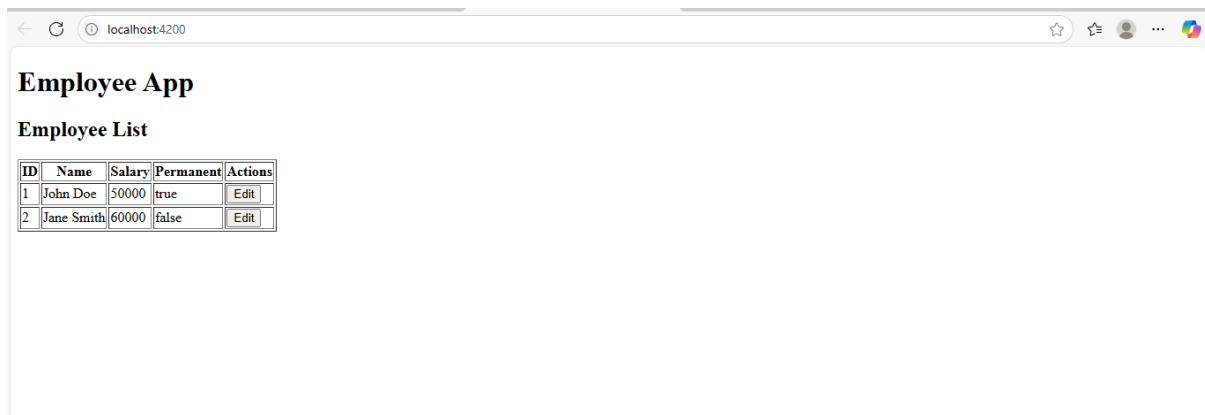
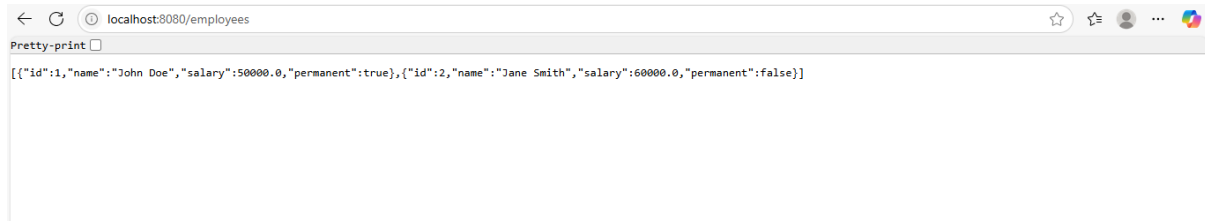
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class Employee {

  constructor() {}
}

```

Output:



Create static employee list data using spring xml configuration

Follow steps below to accomplish this activity:

- Incorporate the following in employee.xml:
 - Create one or two more departments
 - Create four more instances of Employee. (use employee sample data from angular)
 - Reuse existing skills instead of creating new ones
 - Include all four employee instances in an ArrayList.
- In EmployeeDao, incorporate the following:
 - Create static variable with name EMPLOYEE_LIST of type ArrayList<Employee>
 - Include constructor that reads employee list from xml config and set the EMPLOYEE_LIST
 - Create method getAllEmployees() that returns the EMPLOYEE_LIST

Answer:

TestEmployeeDao.java:

package com.cognizant.spring;


```
import com.cognizant.spring.dao.EmployeeDao;
```

```
import com.cognizant.spring.model.Employee;
```

```
public class TestEmployeeDao {
```

```
    public static void main(String[] args) {
```

```
        EmployeeDao dao = new EmployeeDao();
```

```
        for (Employee emp : dao.getAllEmployees()) {
```

```
            System.out.println("Employee: " + emp.getName() + ", Salary: " + emp.getSalary());
```

```
        }
```

```
    }
```

```
}
```

EmployeeDao.java:

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
import java.util.ArrayList;
```

```
public class EmployeeDao {
```

```
    public static ArrayList<Employee> EMPLOYEE_LIST;
```

```
    public EmployeeDao() {
```

```
        ApplicationContext context = new ClassPathXmlApplicationContext("employee.xml");
```

```
        EMPLOYEE_LIST = (ArrayList<Employee>) context.getBean("employeeList");
```

```
    }
```

Department.java:

```
package com.cognizant.spring.model;
```

```
public class Department {
```

```
    private int id;
```

```
    private String name;
```

```
public int getId() { return id; }  
public void setId(int id) { this.id = id; }  
public String getName() { return name; }  
public void setName(String name) { this.name = name; }  
}
```

Employee.java:

```
package com.cognizant.spring.model;
```

```
import java.util.List;
```

```
public class Employee {  
    private int id;  
    private String name;  
    private double salary;  
    private boolean permanent;  
    private Department department;  
    private List<Skill> skills;  
  
    public int getId() { return id; }  
    public void setId(int id) { this.id = id; }  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
    public double getSalary() { return salary; }  
    public void setSalary(double salary) { this.salary = salary; }  
    public boolean isPermanent() { return permanent; }  
    public void setPermanent(boolean permanent) { this.permanent = permanent; }  
    public Department getDepartment() { return department; }  
    public void setDepartment(Department department) { this.department = department; }  
    public List<Skill> getSkills() { return skills; }  
}
```

```

    public void setSkills(List<Skill> skills) { this.skills = skills; }
}

```

Skill.java:

```

package com.cognizant.spring.model;

```

```

public class Skill {
    private int id;
    private String name;

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}

public ArrayList<Employee> getAllEmployees() {
    return EMPLOYEE_LIST;
}
}

```

Employee.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">

    <!-- Departments -->

    <bean id="dept1" class="com.cognizant.spring.model.Department">
        <property name="id" value="1"/>
        <property name="name" value="Human Resources"/>
    </bean>

    <bean id="dept2" class="com.cognizant.spring.model.Department">

```

```
<property name="id" value="2"/>
<property name="name" value="Finance"/>
</bean>
```

```
<!-- Skills -->
<bean id="skill1" class="com.cognizant.spring.model.Skill">
  <property name="id" value="1"/>
  <property name="name" value="Java"/>
</bean>
<bean id="skill2" class="com.cognizant.spring.model.Skill">
  <property name="id" value="2"/>
  <property name="name" value="Angular"/>
</bean>
```

```
<!-- Employees -->
<bean id="emp1" class="com.cognizant.spring.model.Employee">
  <property name="id" value="101"/>
  <property name="name" value="John Doe"/>
  <property name="salary" value="50000"/>
  <property name="permanent" value="true"/>
  <property name="department" ref="dept1"/>
  <property name="skills">
    <list>
      <ref bean="skill1"/>
      <ref bean="skill2"/>
    </list>
  </property>
</bean>
```

```
<bean id="emp2" class="com.cognizant.spring.model.Employee">
  <property name="id" value="102"/>
```

```
<property name="name" value="Jane Smith"/>
<property name="salary" value="60000"/>
<property name="permanent" value="false"/>
<property name="department" ref="dept2"/>
<property name="skills">
    <list>
        <ref bean="skill1"/>
    </list>
</property>
</bean>

<bean id="emp3" class="com.cognizant.spring.model.Employee">
    <property name="id" value="103"/>
    <property name="name" value="Robert Brown"/>
    <property name="salary" value="55000"/>
    <property name="permanent" value="true"/>
    <property name="department" ref="dept1"/>
    <property name="skills">
        <list>
            <ref bean="skill2"/>
        </list>
    </property>
</bean>

<bean id="emp4" class="com.cognizant.spring.model.Employee">
    <property name="id" value="104"/>
    <property name="name" value="Emily Davis"/>
    <property name="salary" value="62000"/>
    <property name="permanent" value="false"/>
    <property name="department" ref="dept2"/>
    <property name="skills">
```

```

    <list>

        <ref bean="skill1"/>

        <ref bean="skill2"/>

    </list>

</property>

</bean>

<!-- Employee List -->

<bean id="employeeList" class="java.util.ArrayList">

    <constructor-arg>

        <list>

            <ref bean="emp1"/>

            <ref bean="emp2"/>

            <ref bean="emp3"/>

            <ref bean="emp4"/>

        </list>

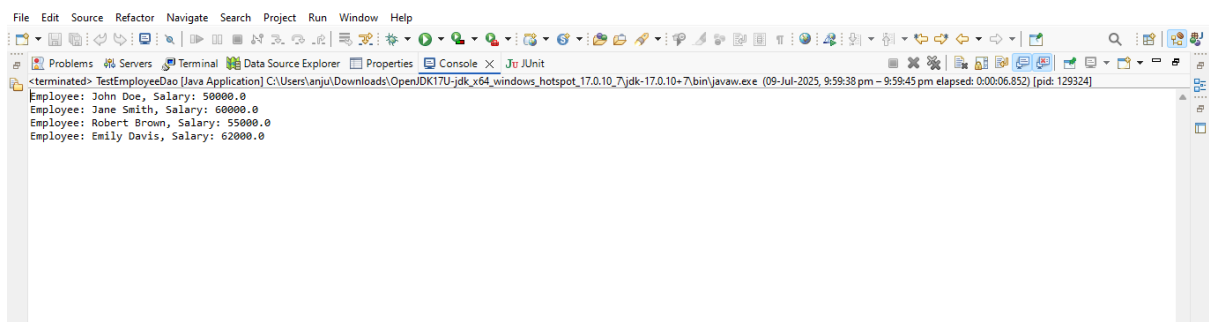
    </constructor-arg>

</bean>

</beans>

```

Output:



```

File Edit Source Refactor Navigate Search Project Run Window Help
<terminated> TestEmployeeDao [Java Application] C:\Users\anjul\Downloads\OpenJDK17U-jdk_x64_windows_hotspot_17.0.10_7-jdk-17.0.10+7\bin\javaw.exe (09-Jul-2025, 9:59:38 pm - 9:59:45 pm elapsed: 0:00:06.852) [pid: 129324]
Employee: John Doe, Salary: 50000.0
Employee: Jane Smith, Salary: 60000.0
Employee: Robert Brown, Salary: 55000.0
Employee: Emily Davis, Salary: 62000.0

```

Create REST service to gets all employees

Follow steps below to accomplish this activity:

- In EmployeeService, incorporate the following:

- Change the annotation for this class from @Component to @Service
 - Create method getAllEmployees() that invokes employeeDao.getAllEmployees() and return the employee list
 - Define @Transactional annotation for this method.
- In EmployeeController, incorporate the following:
 - Include a new get method with name getAllEmployees() that returns the employee list
 - Mark this method as GetMapping annotation with the URL as '/employees'
 - Within this method invoke employeeService.getAllEmployees() and return the same.
 - Test the service using postman.

Answer:

EmployeeserviceApplication.java:

```
package com.example.employeeservice;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
```

```
public class EmployeeserviceApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(EmployeeserviceApplication.class, args);
```

```
    }
```

```
}
```

Employee.java:

```
package com.example.employeeservice.model;
```

```
public class Employee {
```

```
    private int id;
```

```
    private String name;
```

```
    private String designation;
```

```
public Employee() {  
}  
  
public Employee(int id, String name, String designation) {  
    this.id = id;  
    this.name = name;  
    this.designation = designation;  
}  
  
public int getId() {  
    return id;  
}  
public void setId(int id) {  
    this.id = id;  
}  
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}  
public String getDesignation() {  
    return designation;  
}  
public void setDesignation(String designation) {  
    this.designation = designation;  
}  
}
```

EmployeeDao.java:

```
package com.example.employeeservice.dao;
```



```
import com.example.employeeservice.model.Employee;
```

```
import org.springframework.stereotype.Repository;
```

```
import java.util.Arrays;
```

```
import java.util.List;
```

```
@Repository
```

```
public class EmployeeDao {
```

```
    public List<Employee> getAllEmployees() {
```

```
        return Arrays.asList(
```

```
            new Employee(1, "John Doe", "Developer"),
```

```
            new Employee(2, "Jane Smith", "Tester"),
```

```
            new Employee(3, "Alice Johnson", "Manager"),
```

```
            new Employee(4, "Bob Brown", "Analyst")
```

```
        );
```

```
    }
```

```
}
```

EmployeeController.java:

```
package com.example.employeeservice.controller;
```

```
import com.example.employeeservice.model.Employee;
```

```
import com.example.employeeservice.service.EmployeeService;
```

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

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

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

```
import java.util.List;
```

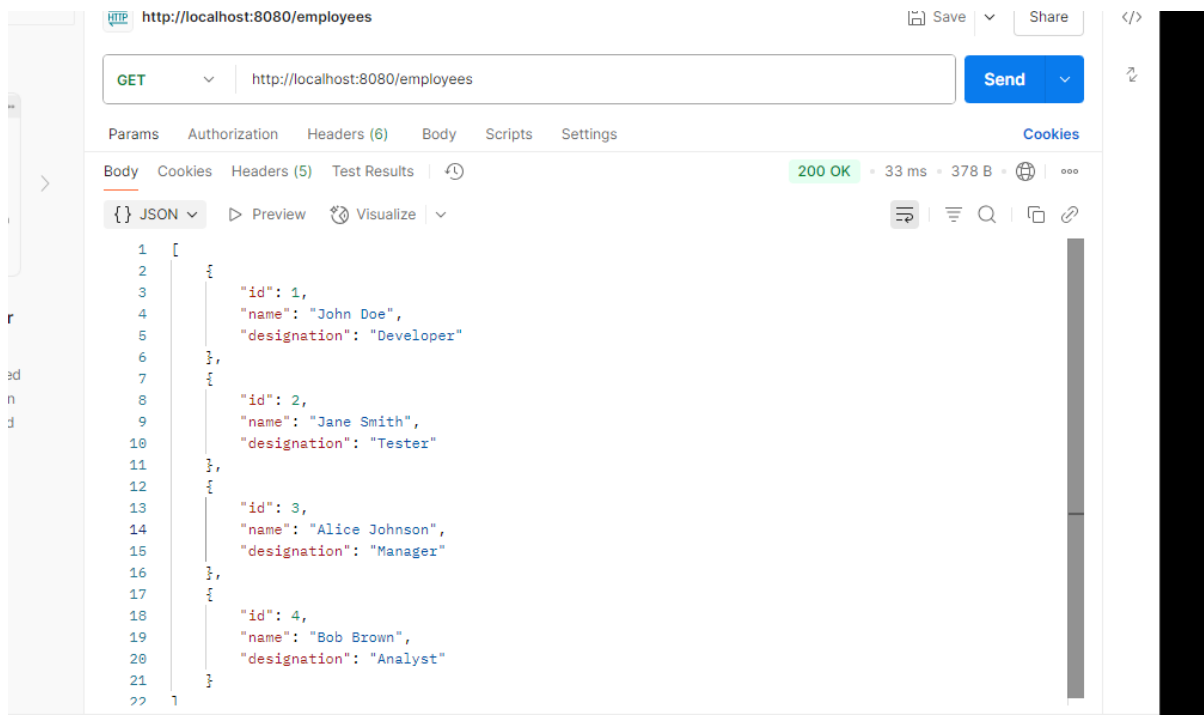
```
@RestController
```

```
public class EmployeeController {  
  
    @Autowired  
    private EmployeeService employeeService;  
  
    @GetMapping("/employees")  
    public List<Employee> getAllEmployees() {  
        return employeeService.getAllEmployees();  
    }  
}
```

EmployeeService.java:

```
package com.example.employeeservice.service;  
  
import com.example.employeeservice.dao.EmployeeDao;  
import com.example.employeeservice.model.Employee;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class EmployeeService {  
  
    @Autowired  
    private EmployeeDao employeeDao;  
  
    public List<Employee> getAllEmployees() {  
        return employeeDao.getAllEmployees();  
    }  
}
```

Output:



Create REST service for department

Create a new service to get all the departments.

Follow steps below to achieve this:

- Create a new REST Service, define below list of classes and respective methods:
 - DepartmentController
 - getAllDepartments() with URL "/departments", this method will return array of departments
 - DepartmentService
 - getAllDepartments()
 - DepartmentDao
 - getAllDepartments() - Create a static variable DEPARTMENT_LIST, this should be populated from spring xml configuration
- Test the service using postman.
- Also verify if department REST service is called by looking into the logs.

Answer:

DemoApplication.java:

package com.example.demo;

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ImportResource;
```

```
@SpringBootApplication
```

```
@ImportResource("classpath:department.xml")
```

```
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

DepartmentController.java:

```
package com.example.demo.controller;
```

```
import com.example.demo.model.Department;
import com.example.demo.service.DepartmentService;
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.RestController;
```

```
import java.util.List;
```

```
@RestController
```

```
public class DepartmentController {
```

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

```
    @Autowired
```

```
private DepartmentService departmentService;
```

```
@GetMapping("/departments")
```

```
public List<Department> getAllDepartments() {  
    LOGGER.info("Department REST service called");  
    return departmentService.getAllDepartments();  
}  
}
```

DepartmentDao.java:

```
package com.example.demo.dao;
```

```
import com.example.demo.model.Department;
```

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

```
import org.springframework.stereotype.Repository;
```

```
import java.util.List;
```

```
@Repository
```

```
public class DepartmentDao {
```

```
@Autowired
```

```
private List<Department> departmentList;
```

```
public List<Department> getAllDepartments() {  
    return departmentList;  
}  
}
```

Department.java:

```
package com.example.demo.model;
```

```
public class Department {
```

```
private String name;
```

```
public Department() {}
```

```
public Department(String name) {  
    this.name = name;  
}
```

```
public String getName() {  
    return name;  
}
```

```
public void setName(String name) {  
    this.name = name;  
}  
}
```

DepartmentService.java:

```
package com.example.demo.service;
```

```
import com.example.demo.dao.DepartmentDao;
```

```
import com.example.demo.model.Department;
```

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

```
import org.springframework.stereotype.Service;
```

```
import java.util.List;
```

```
@Service
```

```
public class DepartmentService {
```

```
    @Autowired
```

```
private DepartmentDao departmentDao;
```

```
public List<Department> getAllDepartments() {  
    return departmentDao.getAllDepartments();  
}  
}
```

Output:

