# **WEEK 3 spring rest using spring boot 3 HANDS ON**

**SUBMITTED BY :-**

NAME – AYUSH SAHOO

SUPERSET ID – 6372243

COURSE – DIGITAL NURTURE 4.0 JAVA FSE

EMAIL – [ayush.sahoo003@gmail.com](mailto:ayush.sahoo003@gmail.com)

COLLEGE – C.V. RAMAN GLOBAL UNIVERSITY

**-----------------NOTE: I HAVE ALSO DONE ADDITIONAL IMPORTANT HANDS ON -------------------**

**MANDATORY HANDS ON**

**(NOTE: THIS IS QUESTION FROM FIRST FILE OF SPRING-REST-HANDSON.DOCX)**

**(THERE WILL BE ANOTHER QUESTIONS FROMM DIIFFERENT FILE OF SPRING-REST-HANDSON.DOCX)**

**EXERCISE 1:**

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

1. **Go to**[**https://start.spring.io/**](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:**

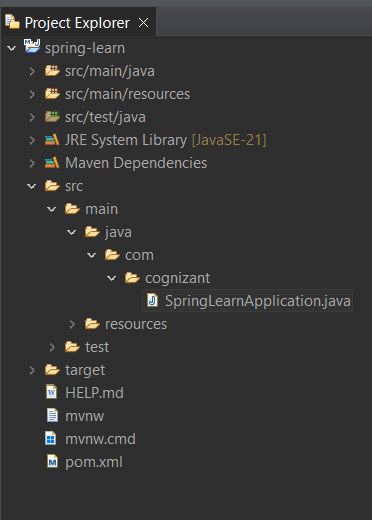
1. **src/main/java - Folder with application code**
2. **src/main/resources - Folder for application configuration**
3. **src/test/java - Folder with code for testing the application**
4. **SpringLearnApplication.java - Walkthrough the main() method.**
5. **Purpose of @SpringBootApplication annotation**
6. **pom.xml**
   1. **Walkthrough all the configuration defined in XML file**
   2. **Open 'Dependency Hierarchy' and show the dependency tree**

**SOLUTION:-**

Step 1:



Step 2: Project Structure



Step 3: Log included in main class of “SpringLearnApplication.java”.

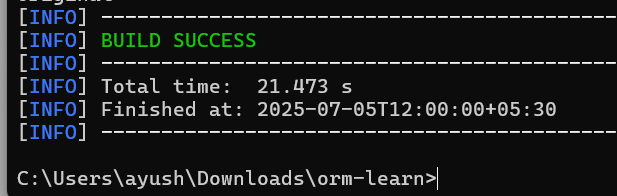


Build with maven in (cmd).

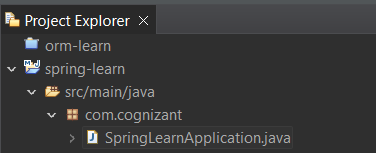
Command: NOTE: here we have used superset id at last instead of random number (mine is 6372243)

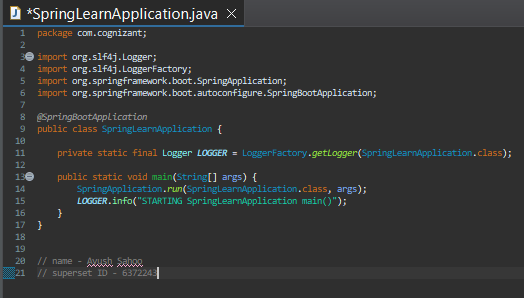


Result:

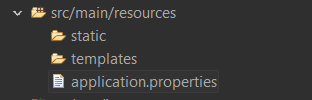


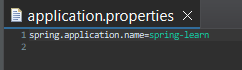
Step 4: src/main/java



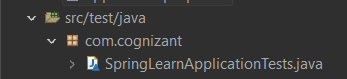


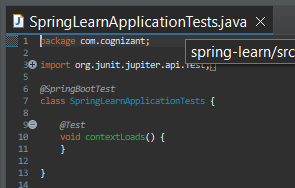
Step 5: src/main/resources





Step 6: src/main/tests





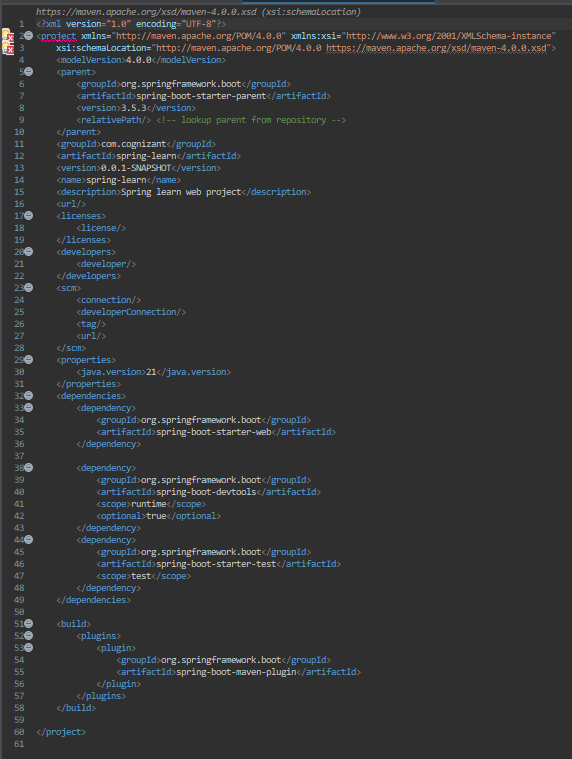
Step 7: Purpose of **@SpringBootApplication** annotations.

This is a combination of three annotations:

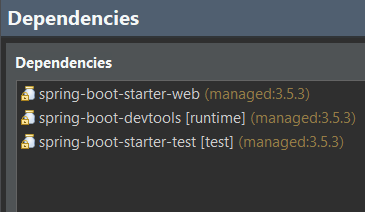
* **@Configuration:** Enables Java-based configuration
* **@EnableAutoConfiguration:** Enables automatic configuration of Spring beans
* **@ComponentScan:** Scans for components in com.cognizant and subpackages

It marks the class as the starting point for the application.

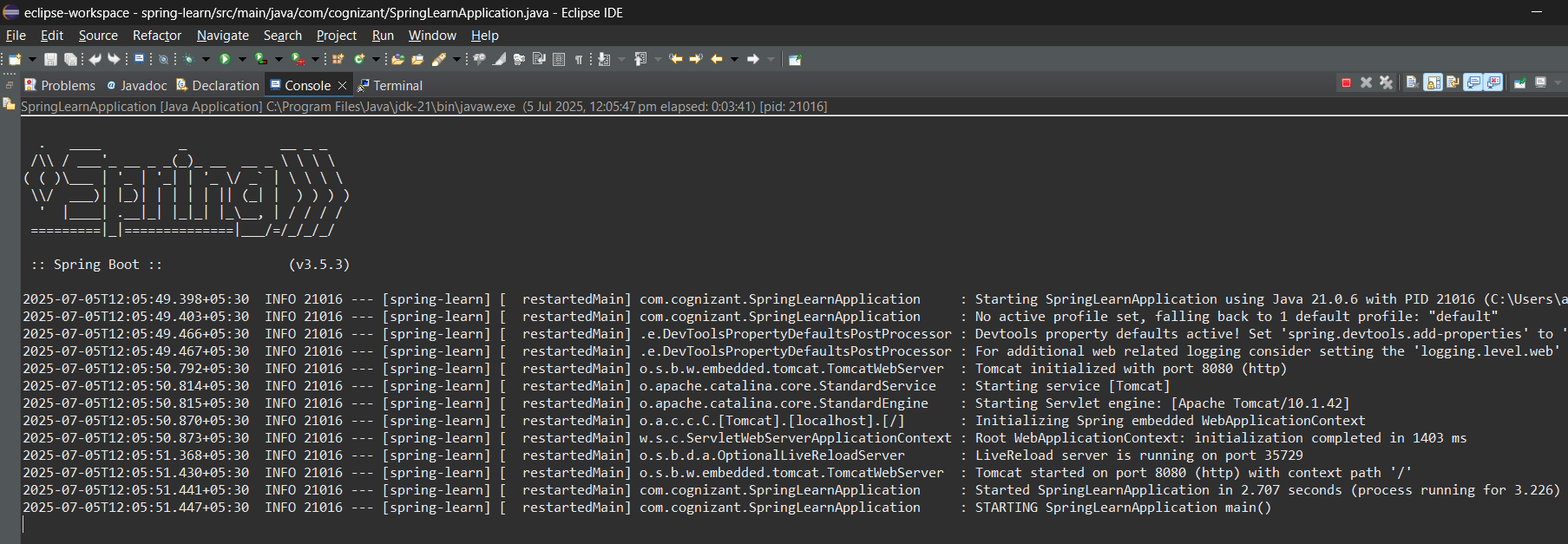
Step 8: “Pom.xml” file configuration.



Dependencies:



**OUTPUT:**

**FULL SCREENSHOT:**

**EXERCISE 4**

**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**

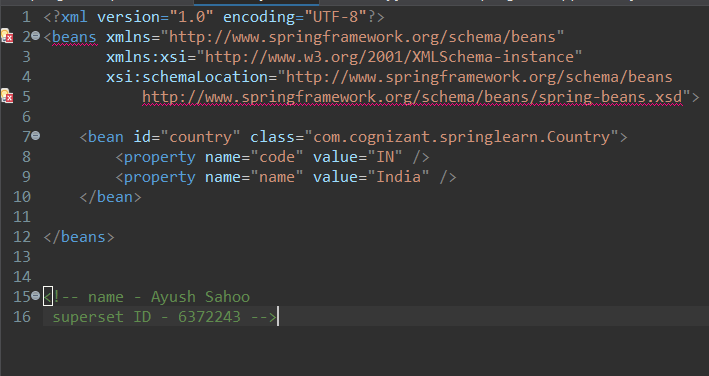
* **Pick any one of your choice country to configure in Spring XML configuration named country.xml.**
* **Create a bean tag in spring configuration for country and set the property and values**
* **Create Country class with following aspects:**
  + **Instance variables for code and name**
  + **Implement 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.**
* **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:**

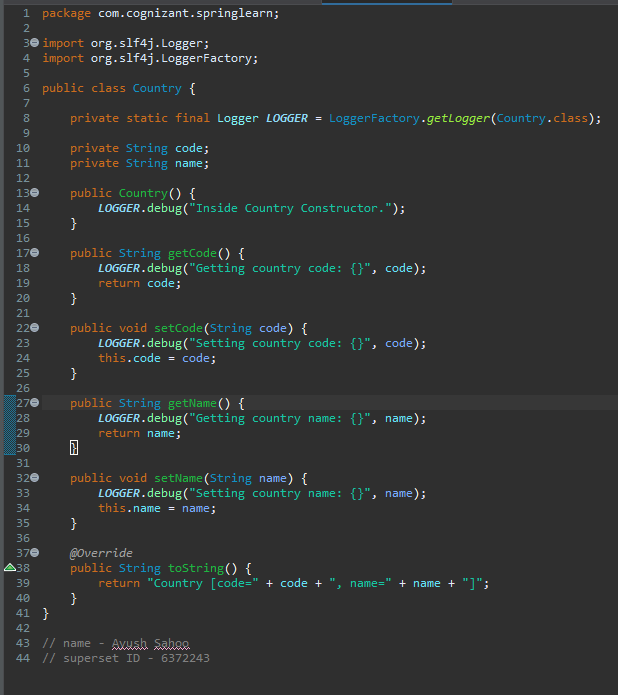
* **bean tag, id attribute, class attribute, property tag, name attribute, value attribute**
* **ApplicationContext, ClassPathXmlApplicationContext**
* **What exactly happens when context.getBean() is invoked**

**SOLUTION:**

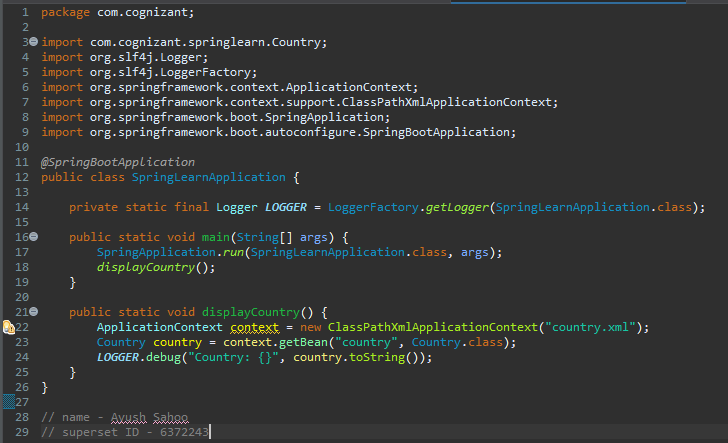
Step 1: in “country.xml” under src/main/resources.



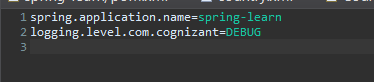
Step 2: In “Country.java” under src/main/java/main/cognizant/springlearn.



Step 3: In “SpringLearnApplication.java “ under src/main/java/com/cognizant.



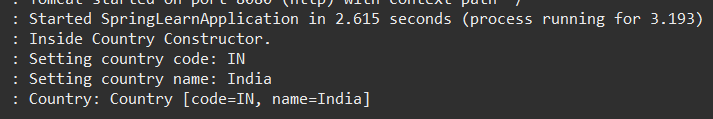
Step 4: In “application.properties” under resources folder.



(NOTE: here we have done debugging as the spring boot application wasn’t working accordingly)

Step 6: Run the application via “SpringLearnApplication”

**OUTPUT:**

****

**FULL SCREENSHOT:**

****

**SME EXPLANATION:**

1. Spring XML Elements

| Element | Description |
| --- | --- |
| <bean> | Defines a Spring-managed object (bean) |
| id | Unique identifier for the bean |
| class | The full path of the class whose object we want to create |
| <property> | Sets property value via setter injection |
| name | Property name (must match variable name).  Name of the variable (like code, name) |
| value | Value to assign to the property.  The actual value to give (like "IN" or "India") |

3. ApplicationContext vs ClassPathXmlApplicationContext

In Spring, **ApplicationContext** is the heart of the system. It manages the beans (objects), connects them, and gives them to us when needed.

* ApplicationContext is the central Spring container interface
* ClassPathXmlApplicationContext is a concrete implementation that loads beans from a \*.xml file located in src/main/resources

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

3.What Happens When context.getBean() is Invoked?

When we write : Country country = context.getBean("country", Country.class);

* Spring reads the XML config
* Instantiates the class defined in <bean>
* Injects values into the fields using <property>
* Returns the fully initialized bean object

**--------------------------------------------------------------------------------------------------------------------------------------**

**(NOTE: THIS IS ANOTHER QUESTION FROM ANOTHER FILE 2ND FILE OF SPRING-REST-HANDSON.DOCX)**

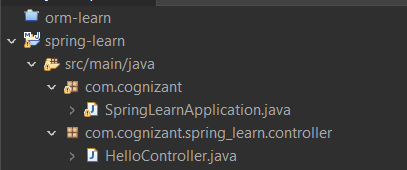
**EXERCISE 2**

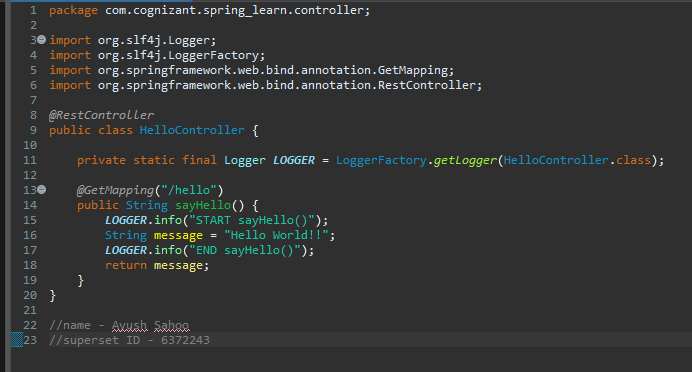
**Hello World RESTful Web Service   
  
Write a REST service in the spring learn application created earlier, that returns the text "Hello World!!" using Spring Web Framework. Refer details below:  
  
Method: GET  
URL: /hello  
Controller: com.cognizant.spring-learn.controller.HelloController  
Method 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:**

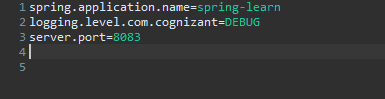
* **In network tab of developer tools show the HTTP header details received**
* **In postman click on "Headers" tab to view the HTTP header details received**

**SOLUTION:**

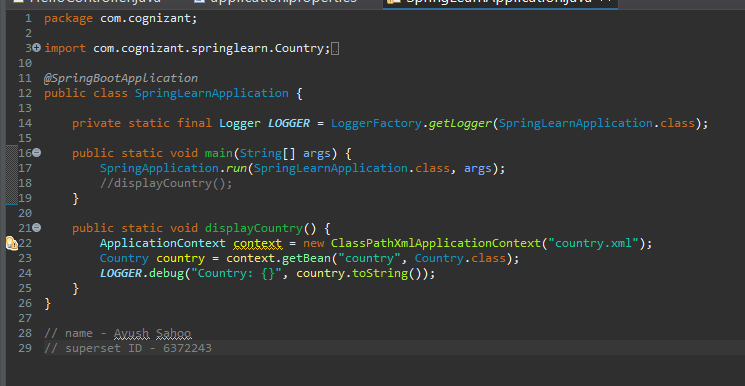
Step 1: Create a package “com.cognizant.spring\_learn.controller” and then create a class under same package by the name of “HelloController.java”.



Step 2: In “application.properties”



Step 3: In “SpringLearnApplication.java”

****

**OUTPUT:**

1. From browser.

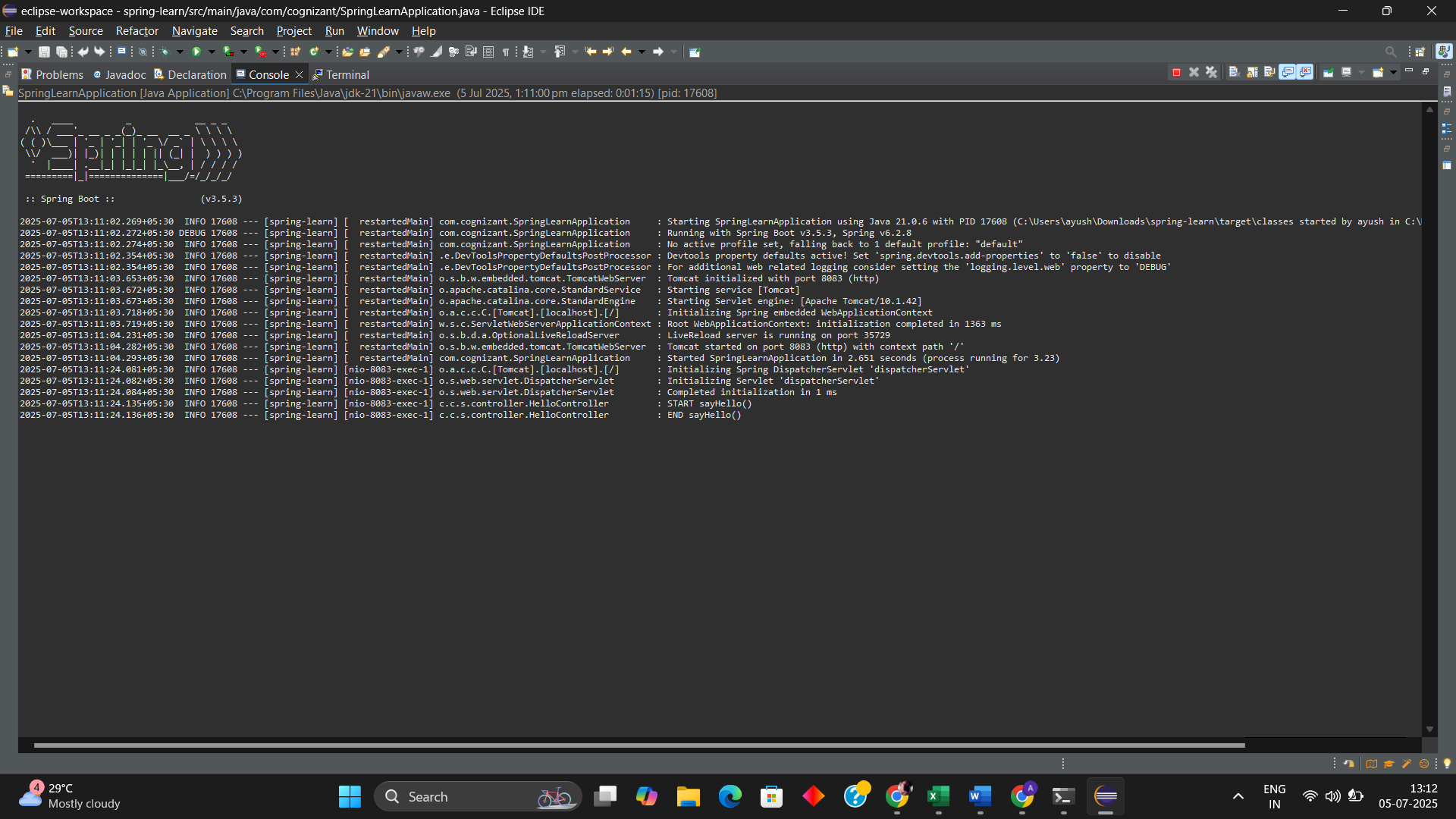
This is from console (eclipse) before running the URL in chrome.

****

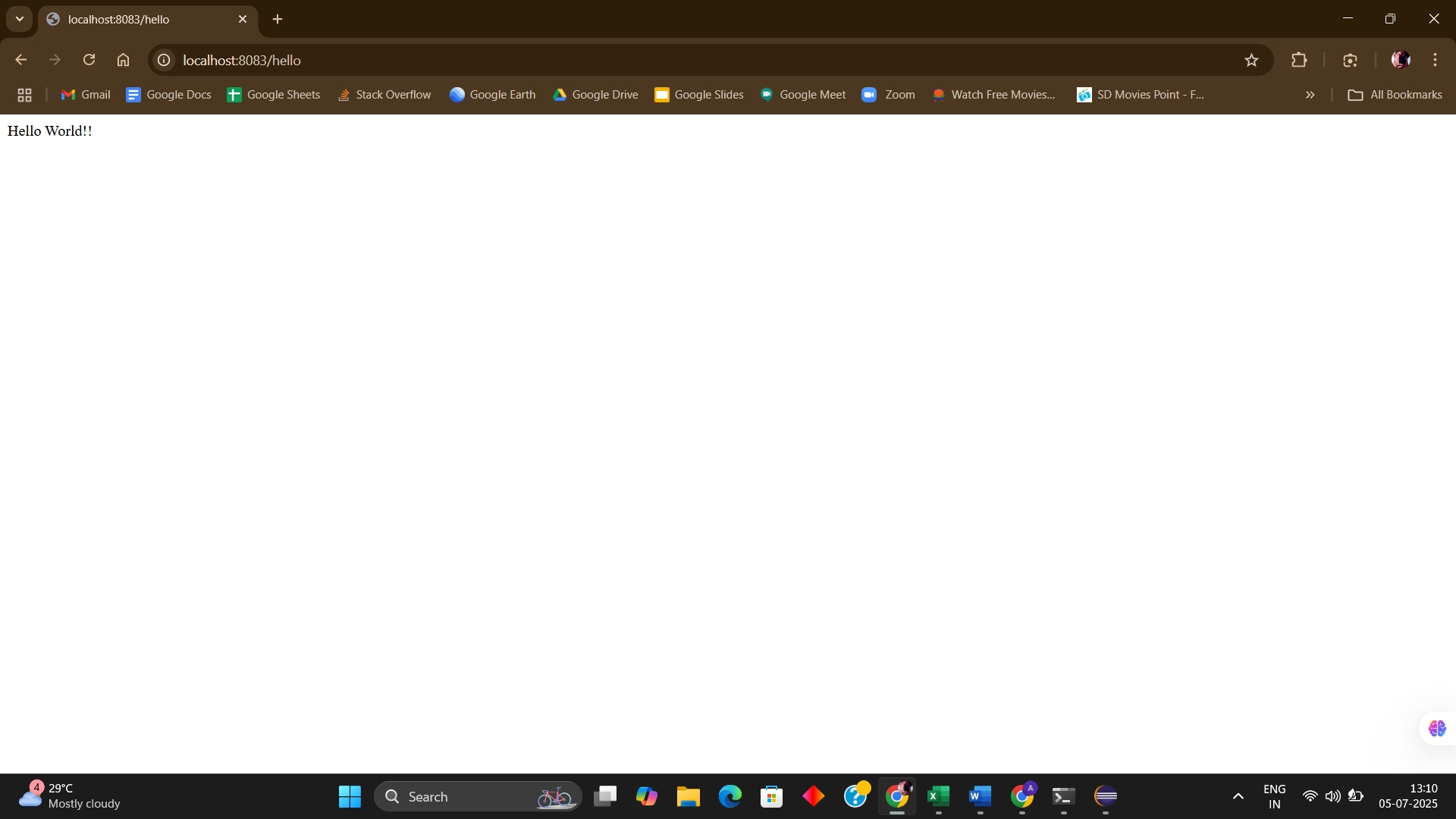
This is after running the URL in chrome**.**

****

**FULL SCREENSHOT:** After running the whole process.

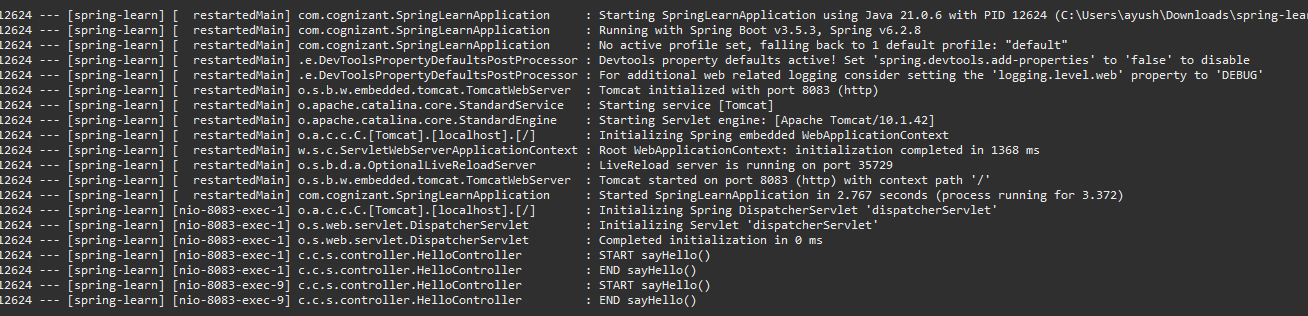
****

**OUTPUT IN BROWSER.**

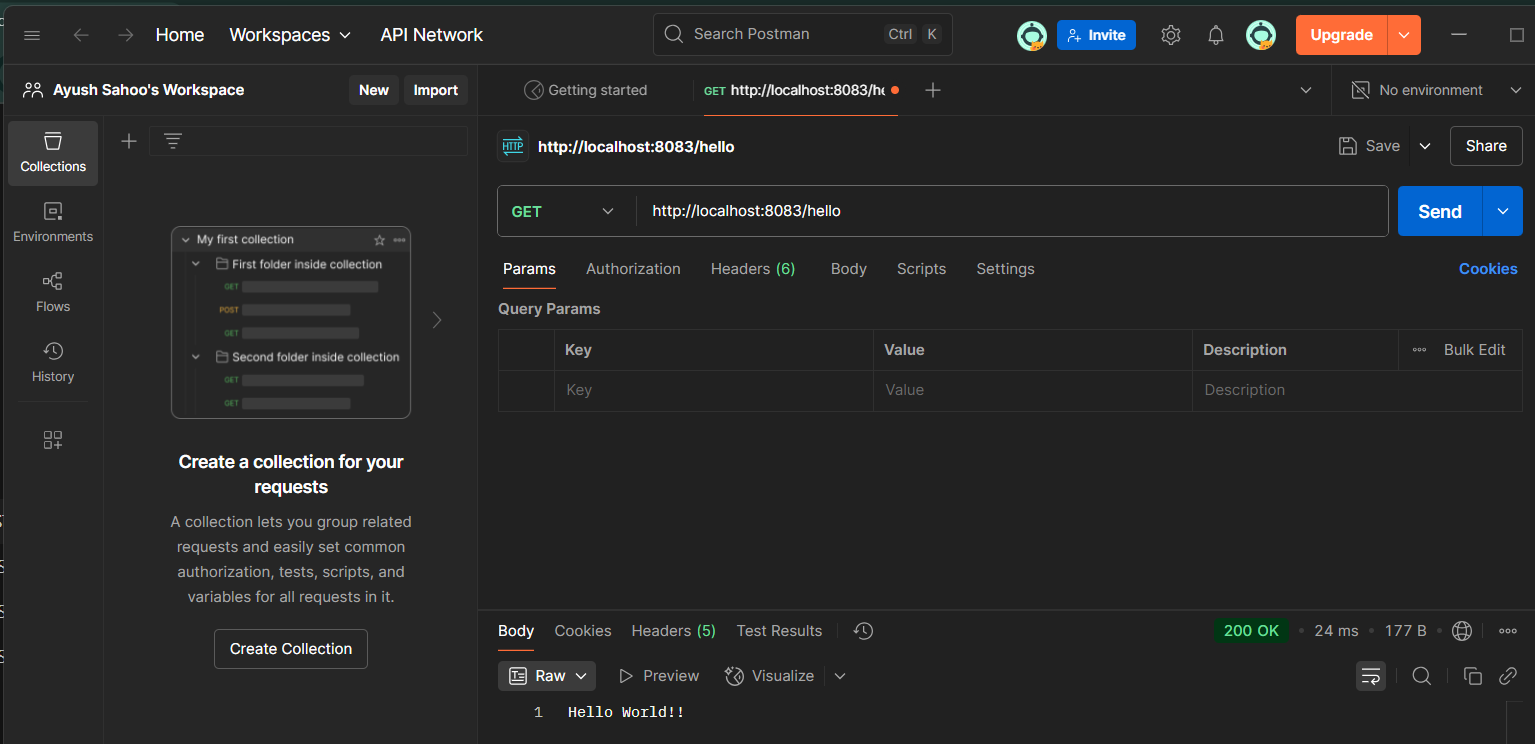
****

1. FROM POSTMAN

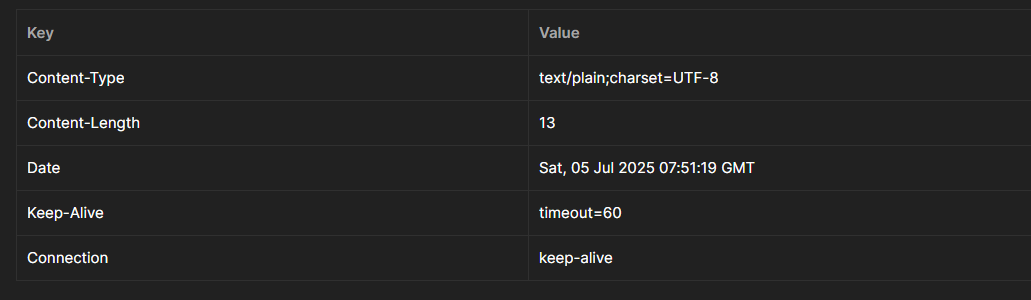
Whole screenshot after completing the process.

****

**OUTPUT from POSTMAN.**

****

**Headers section from POSTMAN.**

****

**Step 4:**

- **Content-Type:** Describes type of response (text/plain in our case)

- **Content-Length:** The length of the response body in bytes (13 for "Hello World!!")

- **Date:** Server response time

- **Keep-Alive and Connection**: Manage connection behavior

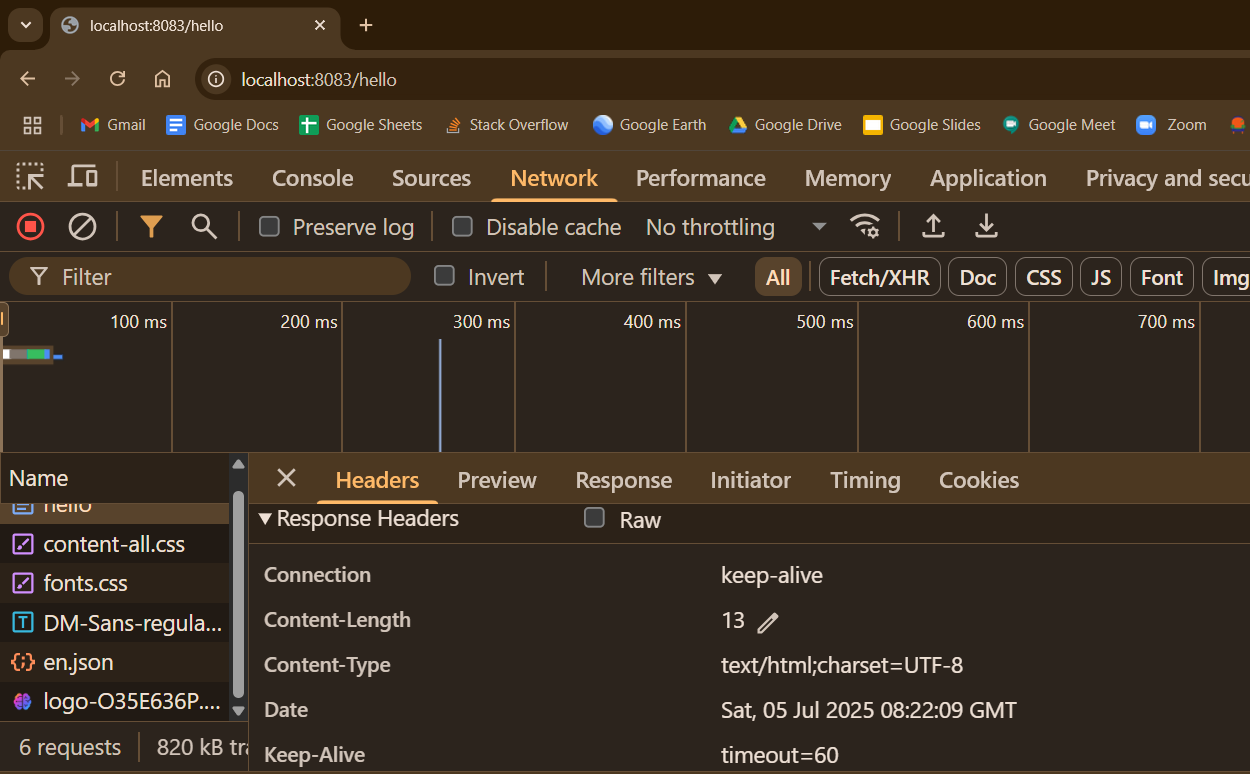
THESE ARE THE LOGS.

INFO ... START sayHello()

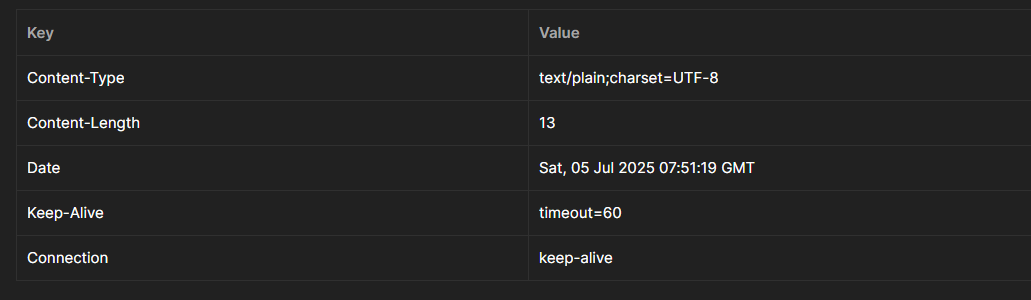
INFO ... END sayHello()

**Step 5: Headers Section from Browser and from POSTMAN**

**HEADER section from Browser**

****

**HEADERS section from POSTMAN.**

****

Note: 'Transfer-Encoding: chunked' was not shown because the content length was explicitly set.

**EXERCISE 3:**

**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"**

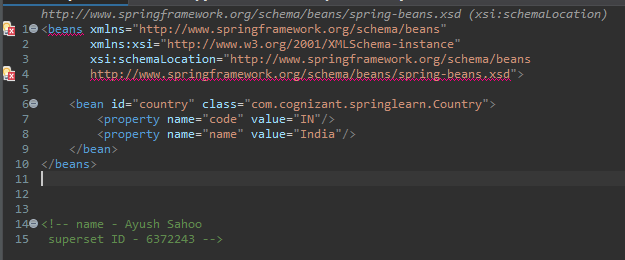
**}**

**SME to explain the following aspects:**

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

**SOLUTION:**

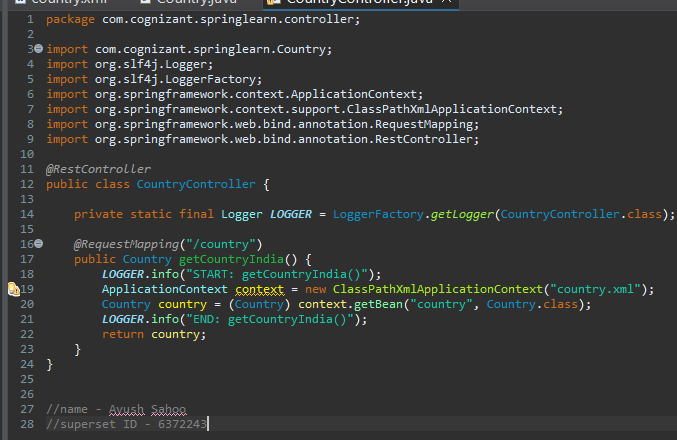
Step 1: In “country.xml” under src/main/resources.



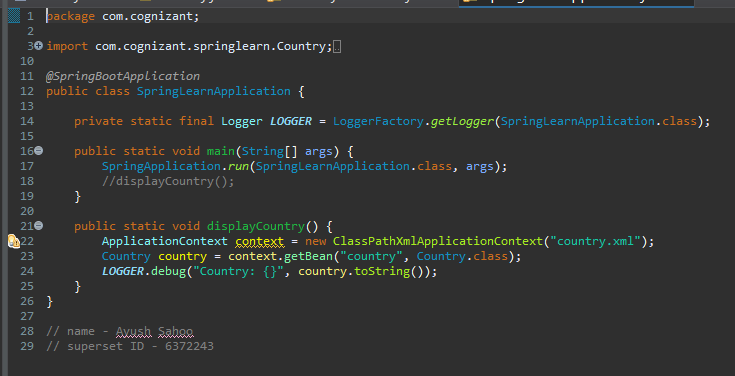
Step 2: In “Country.java” under src/main/java/com/cognizant/springlearn.



Step 3: Created “CountryController.java” under src/main/java/com/cognizant/springlearn/controller.

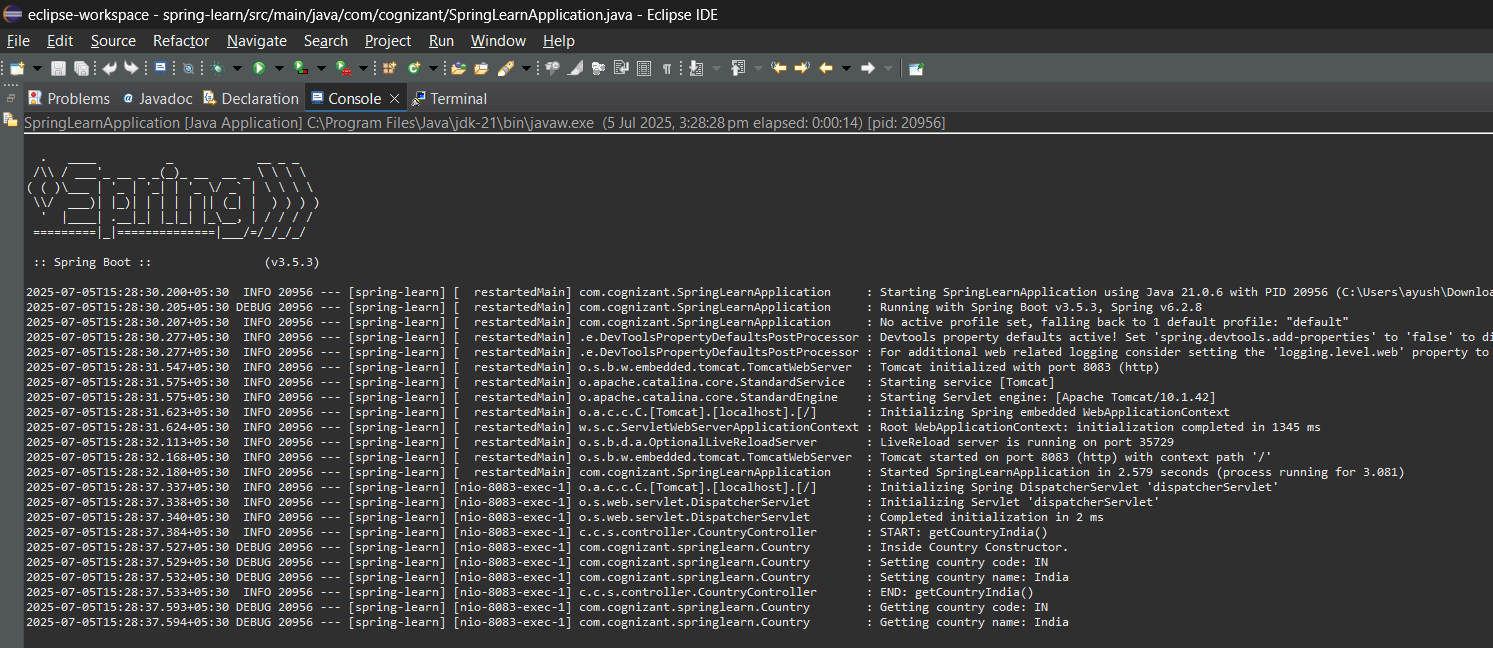


Step 4: Run the application in “SpringLearnApplication.java”



**OUTPUT:**

**Console screen shot**

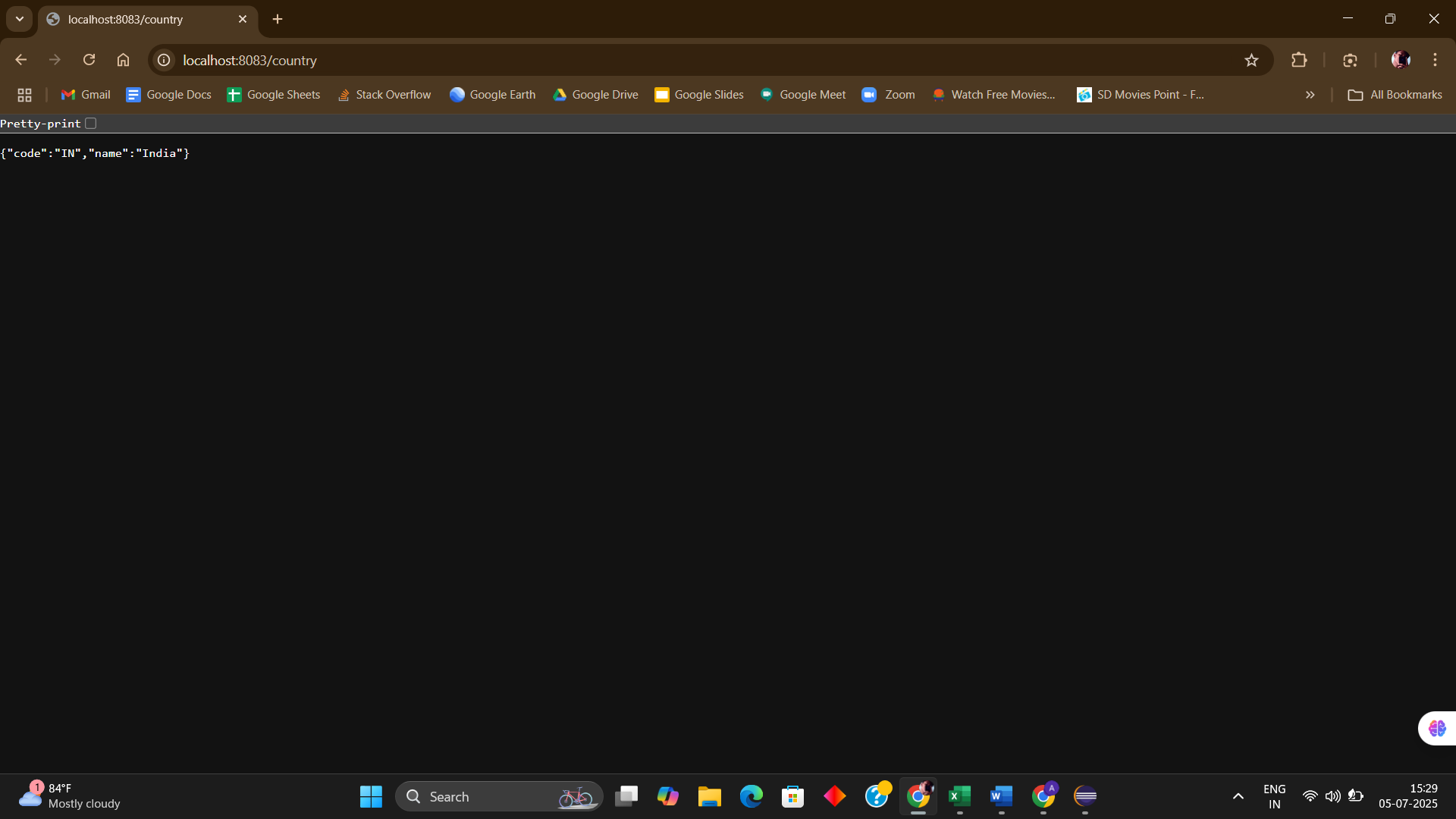
****

Before running web browser URL the console showed

Tomcat started on port 8083

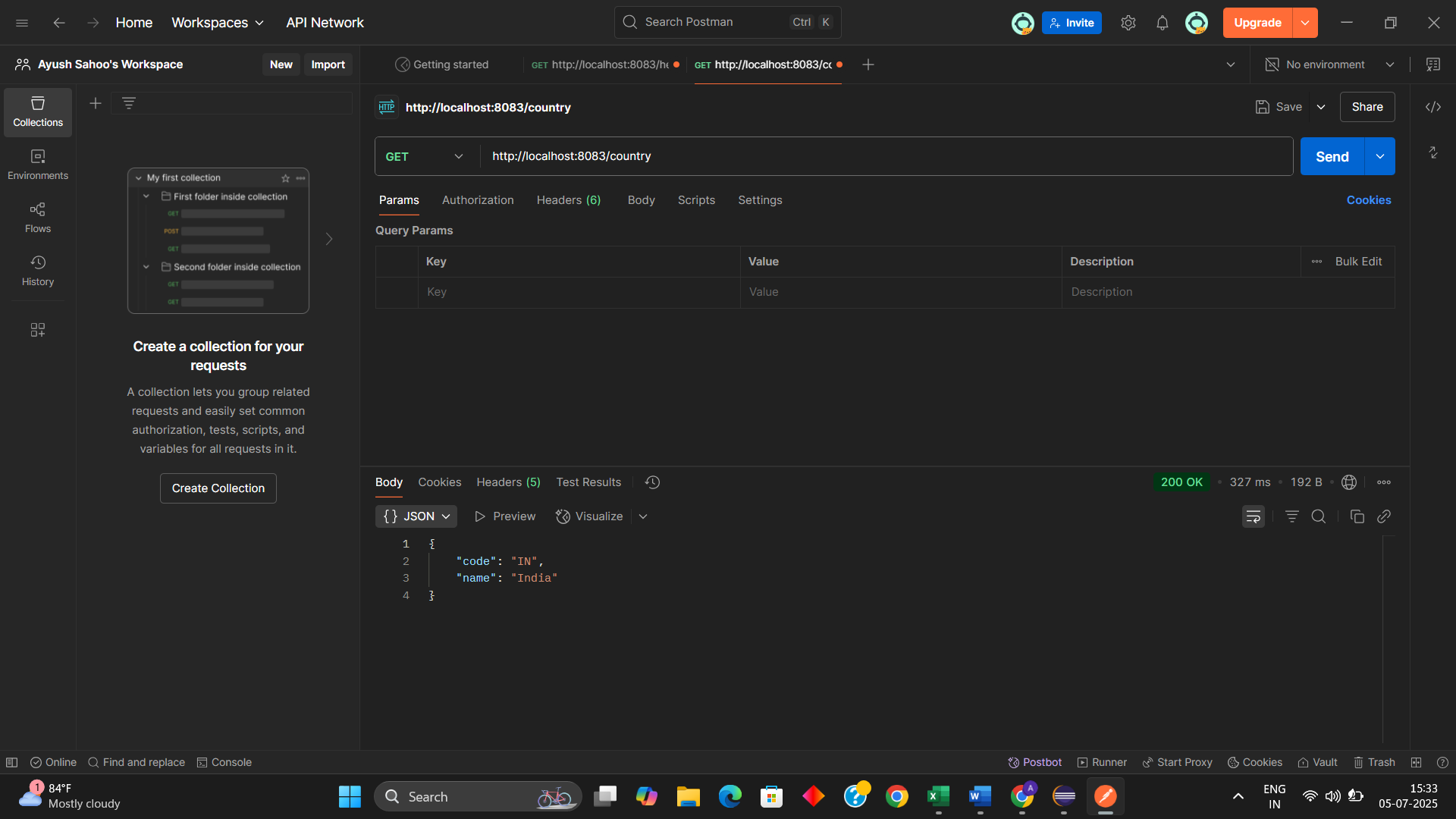


1. **Web browser screen shot and headers tab**

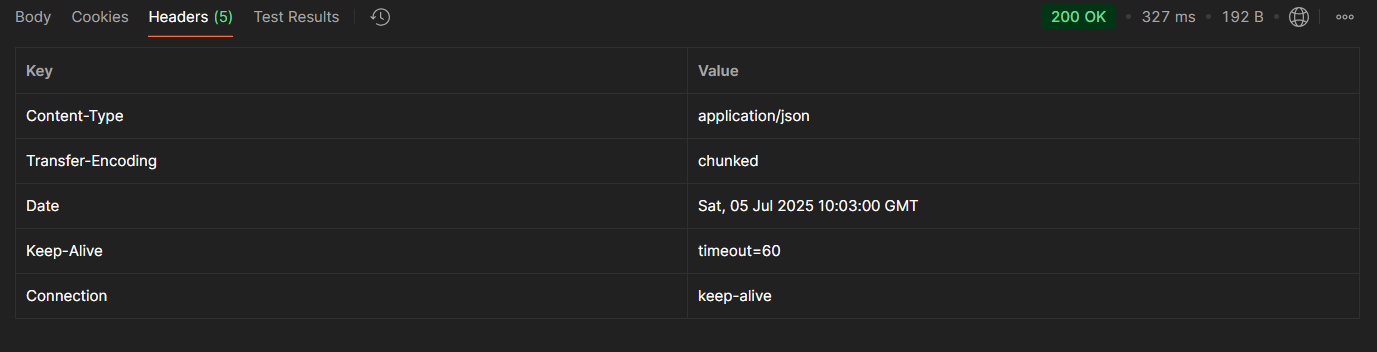
****

****

1. **From POSTMAN and HEADERS Tab**

****

**Headers Tab:**

****

**SME QUESTIONS:**

* **What happens in the controller method?**

ANS. Spring calls getCountryIndia() → loads bean from country.xml → returns as JSON using Jackson.

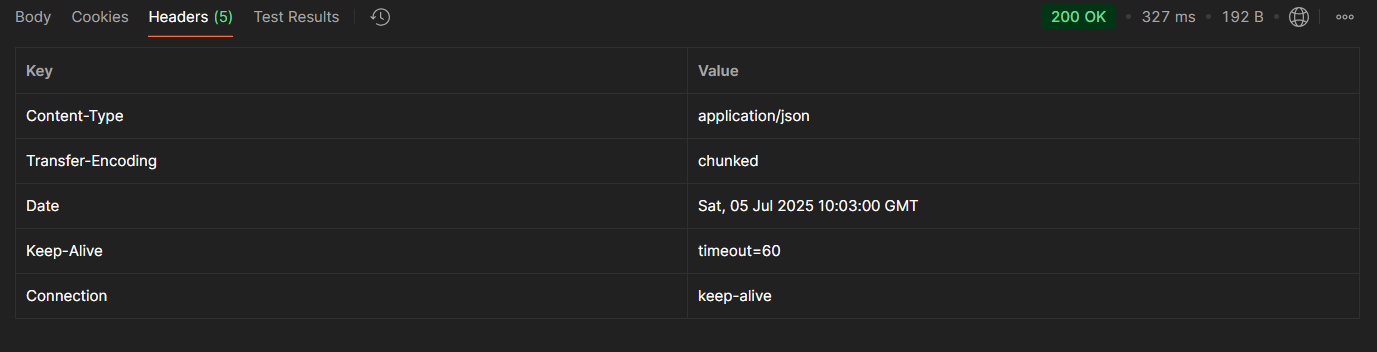
* **How the bean is converted into JSON reponse?**

ANS. Spring Boot uses Jackson (in background) to convert the Country object into JSON format automatically.

* **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**

****

**EXERCISE 4:**

**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:**

**{**

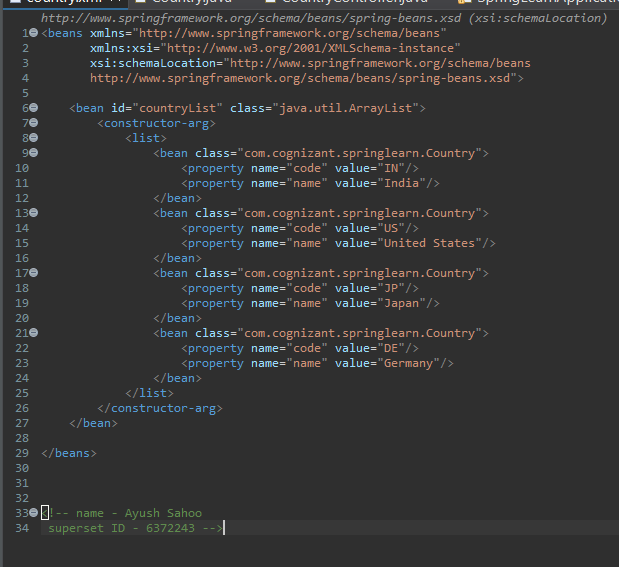
**"code": "IN",**

**"name": "India"**

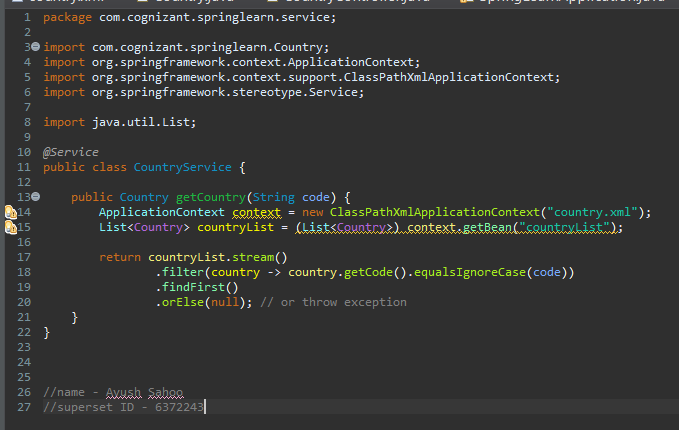
**}**

**SOLUTION:**

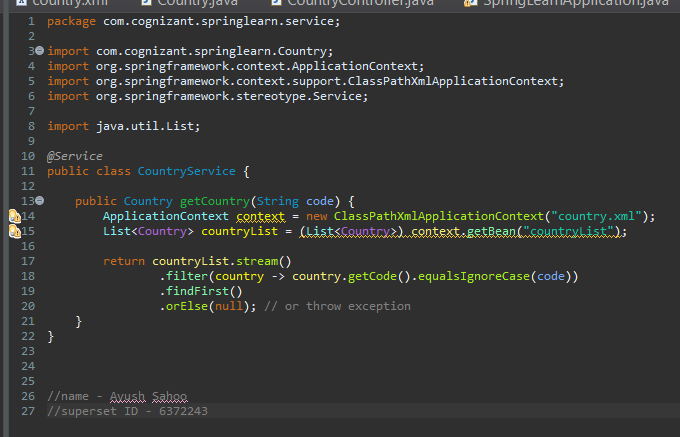
Step 1: In “country.xml” under src/main/resources/country.xml.



Step 2: Create “CountryService.java” under src/main/java/com/cognizant/springlearn/service/CountryService.java.



Step 3: Update the “CountryController.java” under src/main/java/com/cognizant/springlearn/controller.



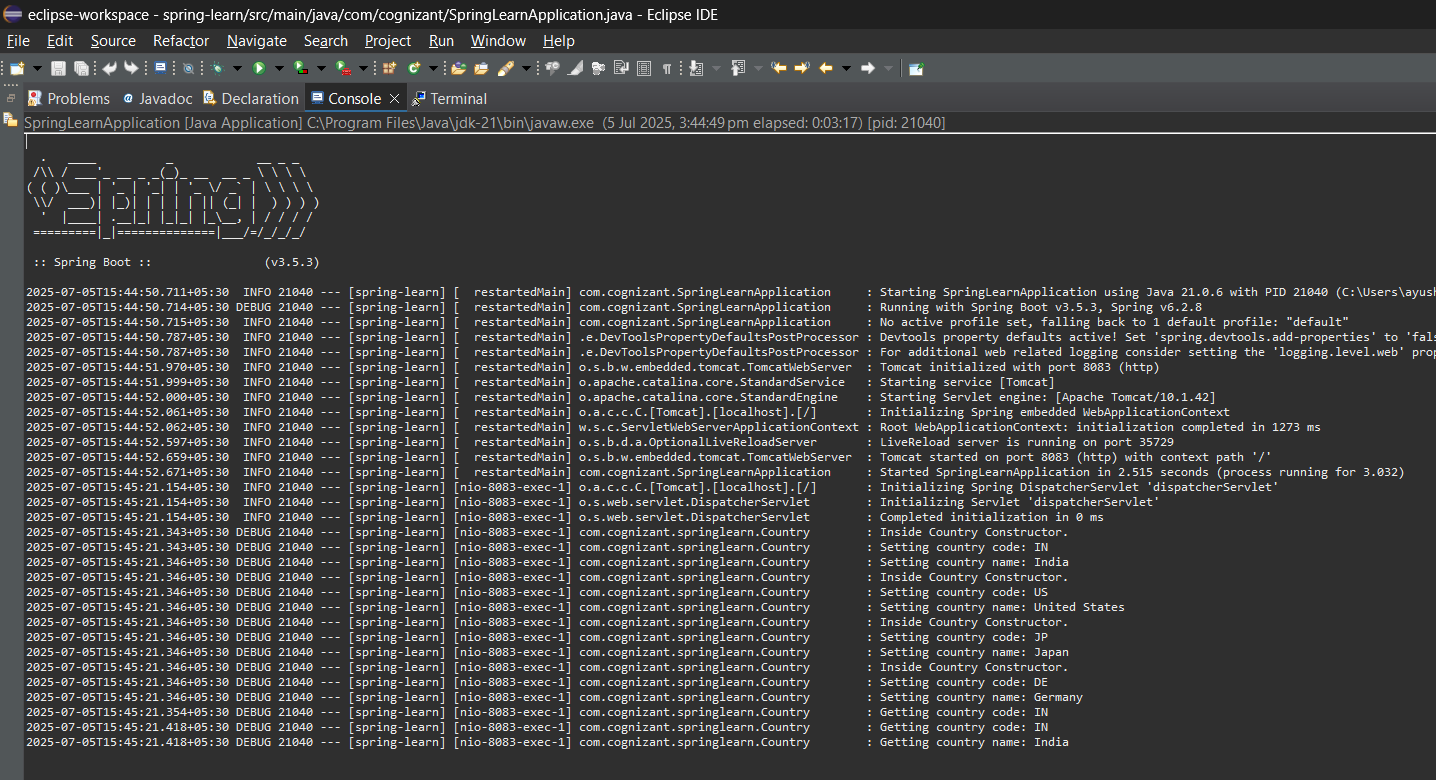
Step 4: Run under “SpringLearnApplication.java”

And the console output will be before opening in browser or POSTMAN.



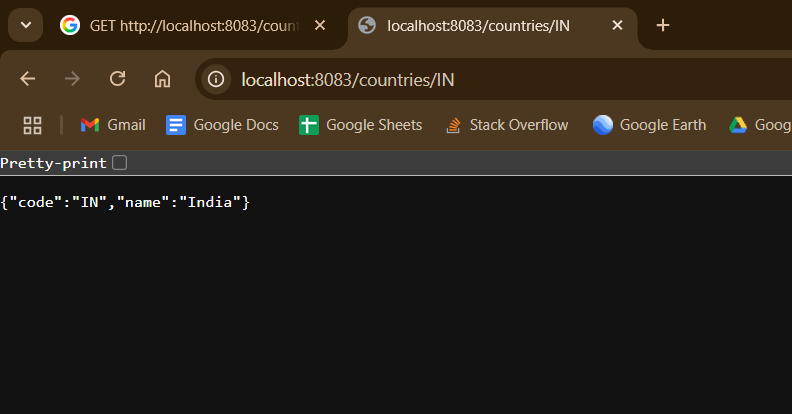
And after running in Web Browser or POSTMAN

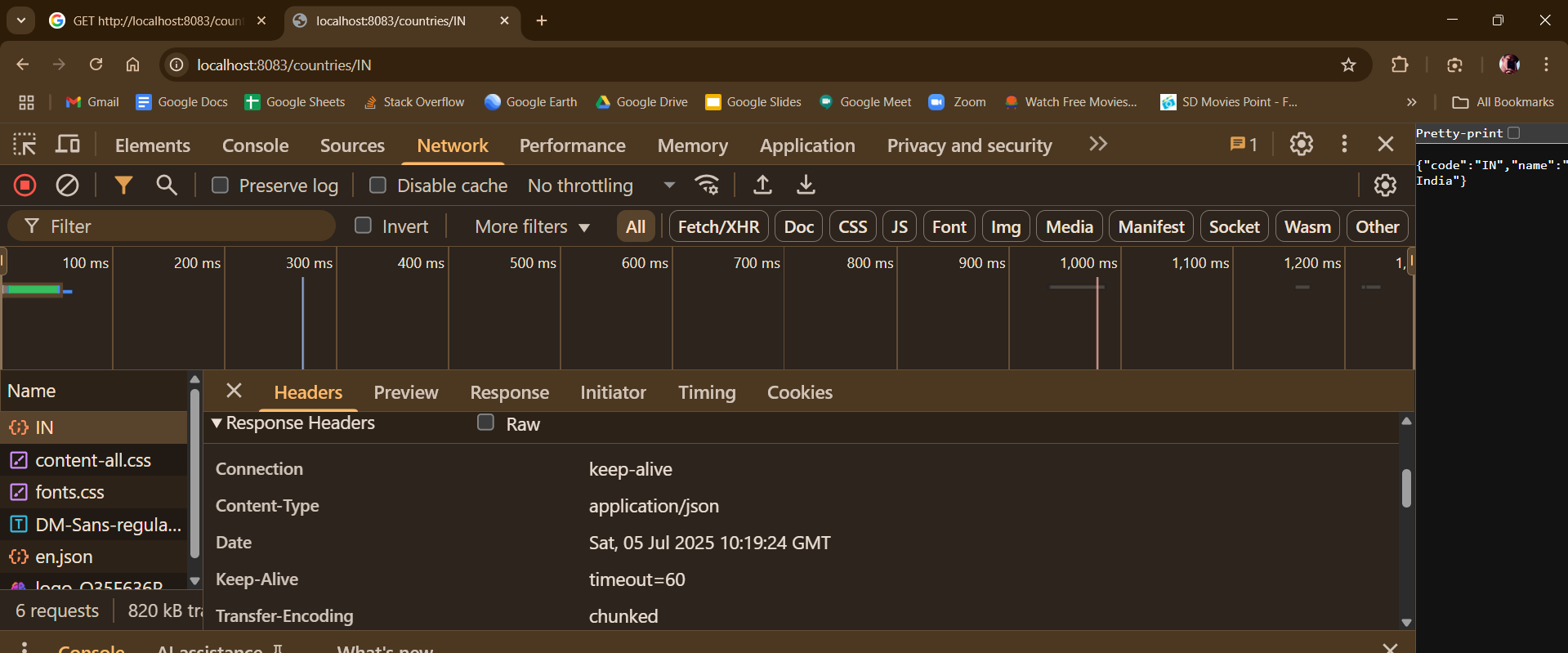
(Full Screenshot)



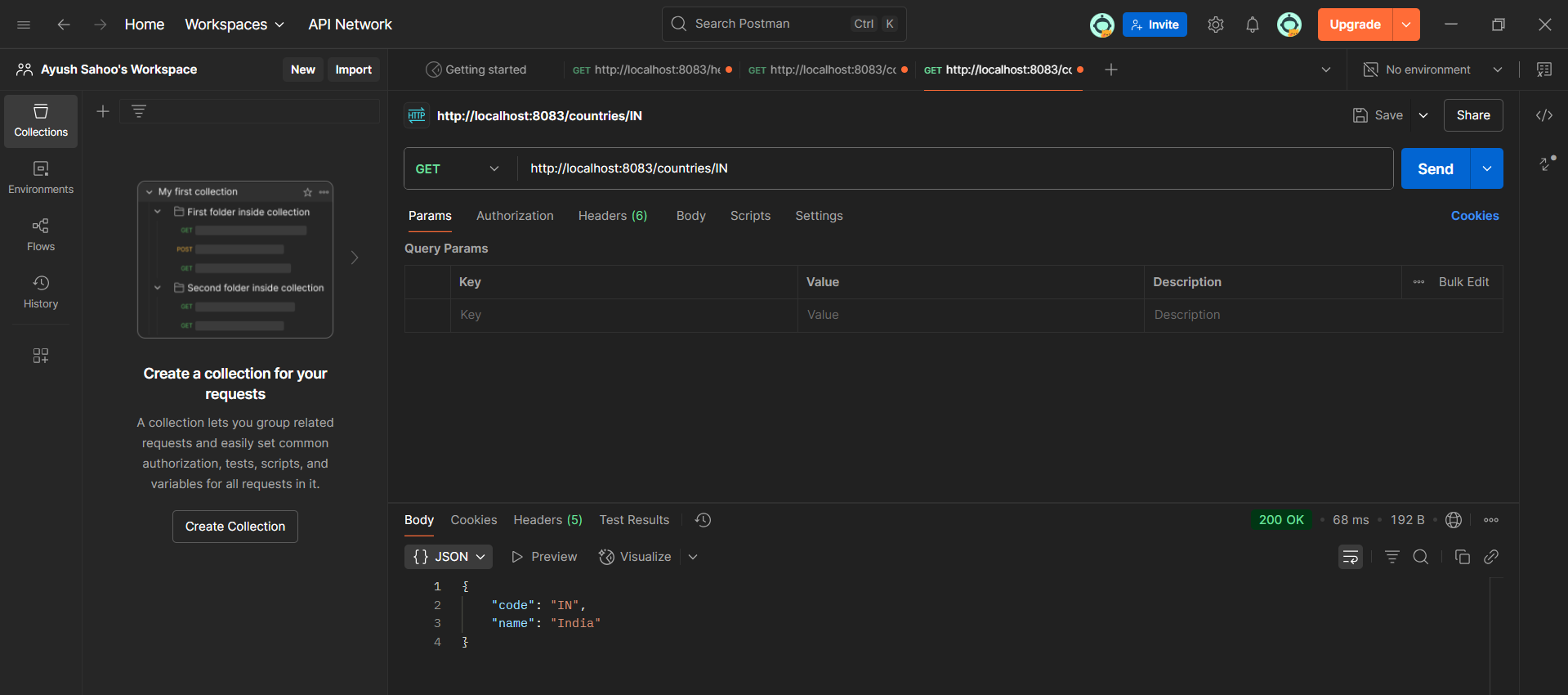
**OUTPUT:**

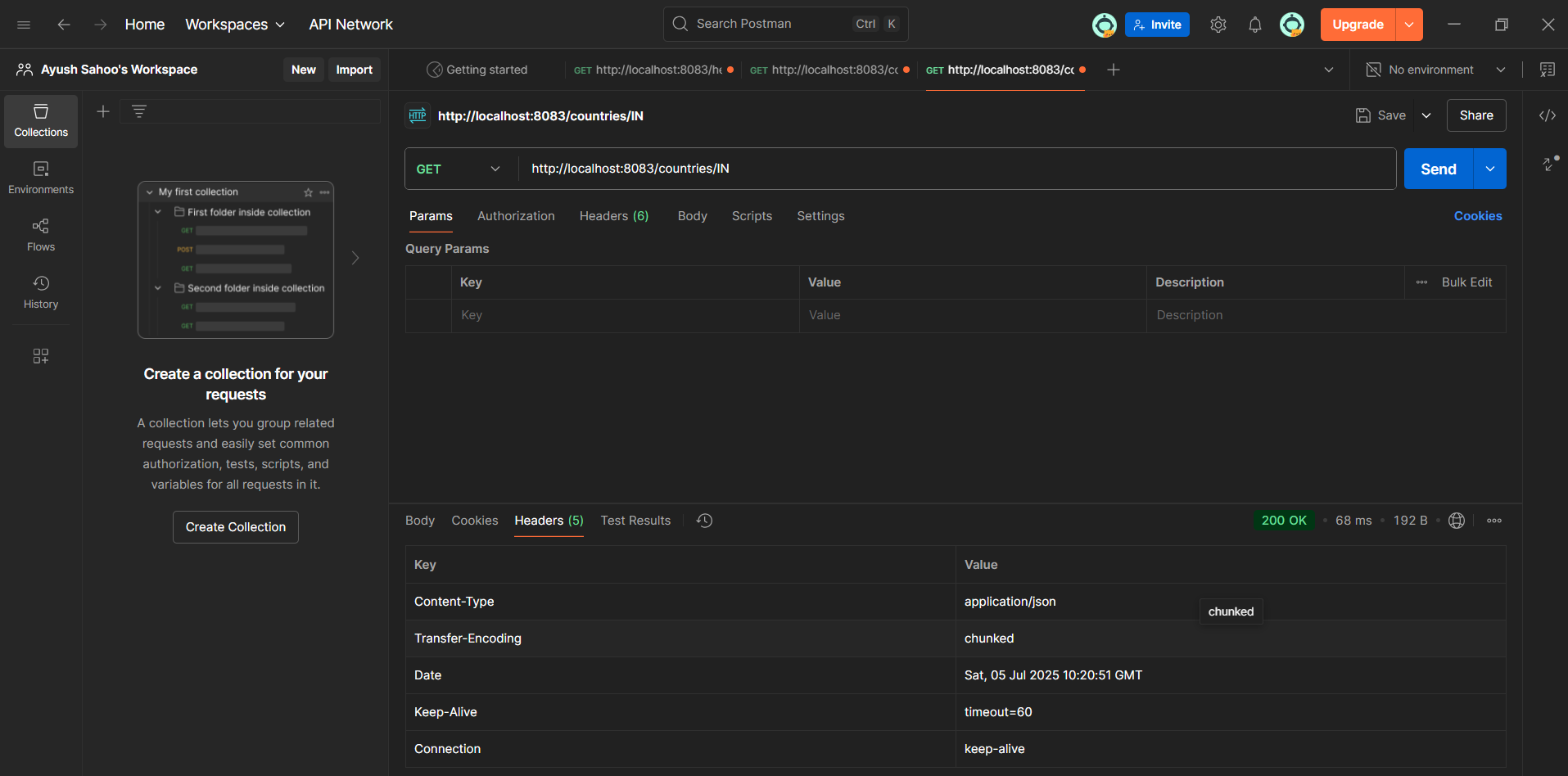
1. **Browser Output and HEADERS Tab.**





1. **POSTMAN Output and HEADERS Tab.**



****

**(NOTE: THIS IS ANOTHER QUESTION FROM ANOTHER FILE OF JWT-HANDSON.DOCX)**

**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**

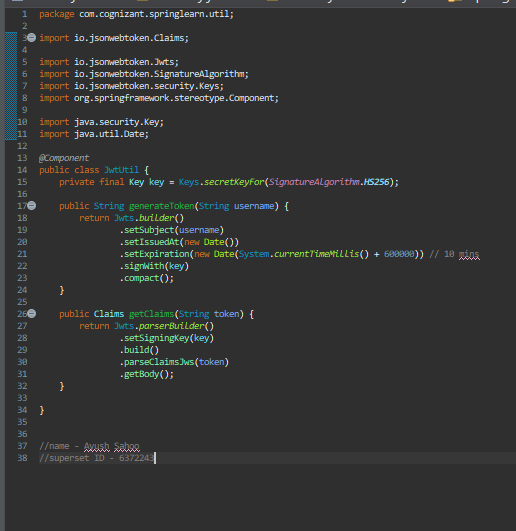
**Let incorporate the above as separate hands on exercises.**

**SOLUTION:**

Step 1: Create “JwtRequestFilter.java” under src/main/java/com/cognizant/springlearn/filter.



Step 2: Create “JwtUtil.java” under src/main/java/com/cognizant/springlearn/util.

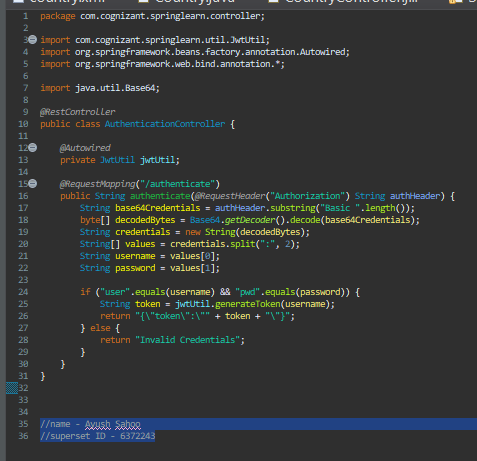


Step 3: Added dependencies in “pom.xml”.

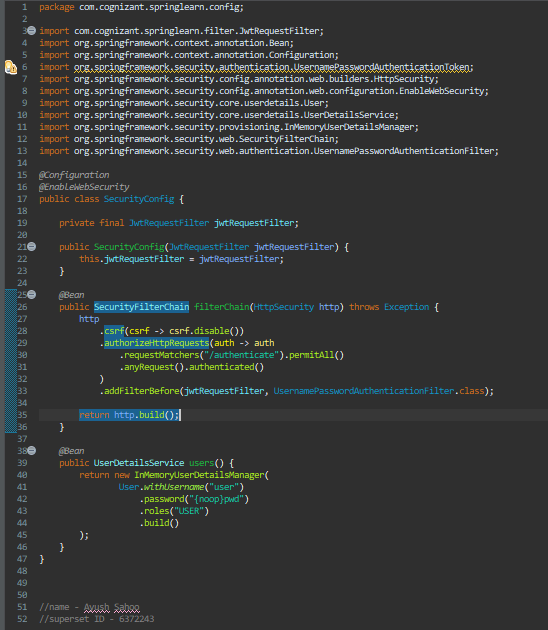


(NOTE: THIS DEPENDENCIES ONLY GOT ADDED INTO OTHER CODE/DEPENDENCIES)

Step 4: Create “AunthenticationController.java” under src/main/java/com/cognizant/springlearn/controller.

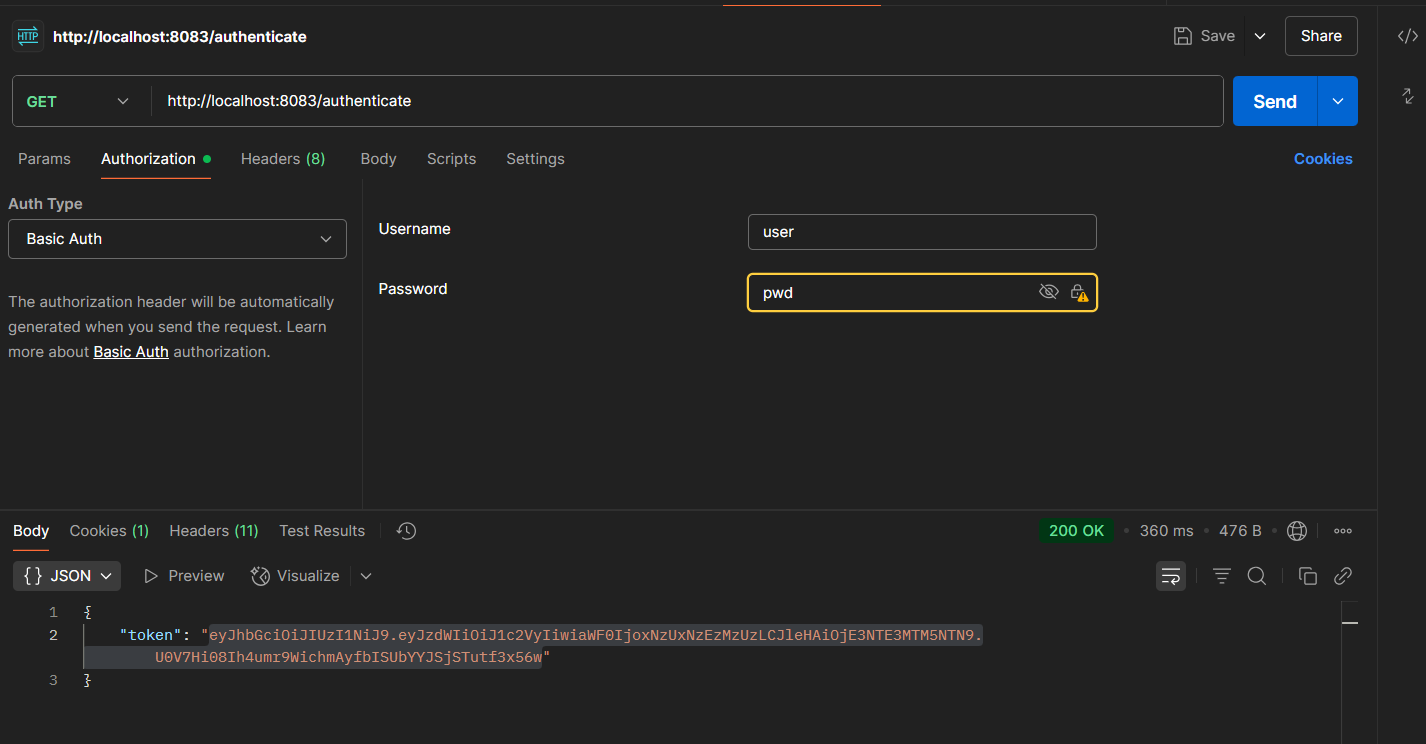


Step 5: Update “SecurityConfig.java”.

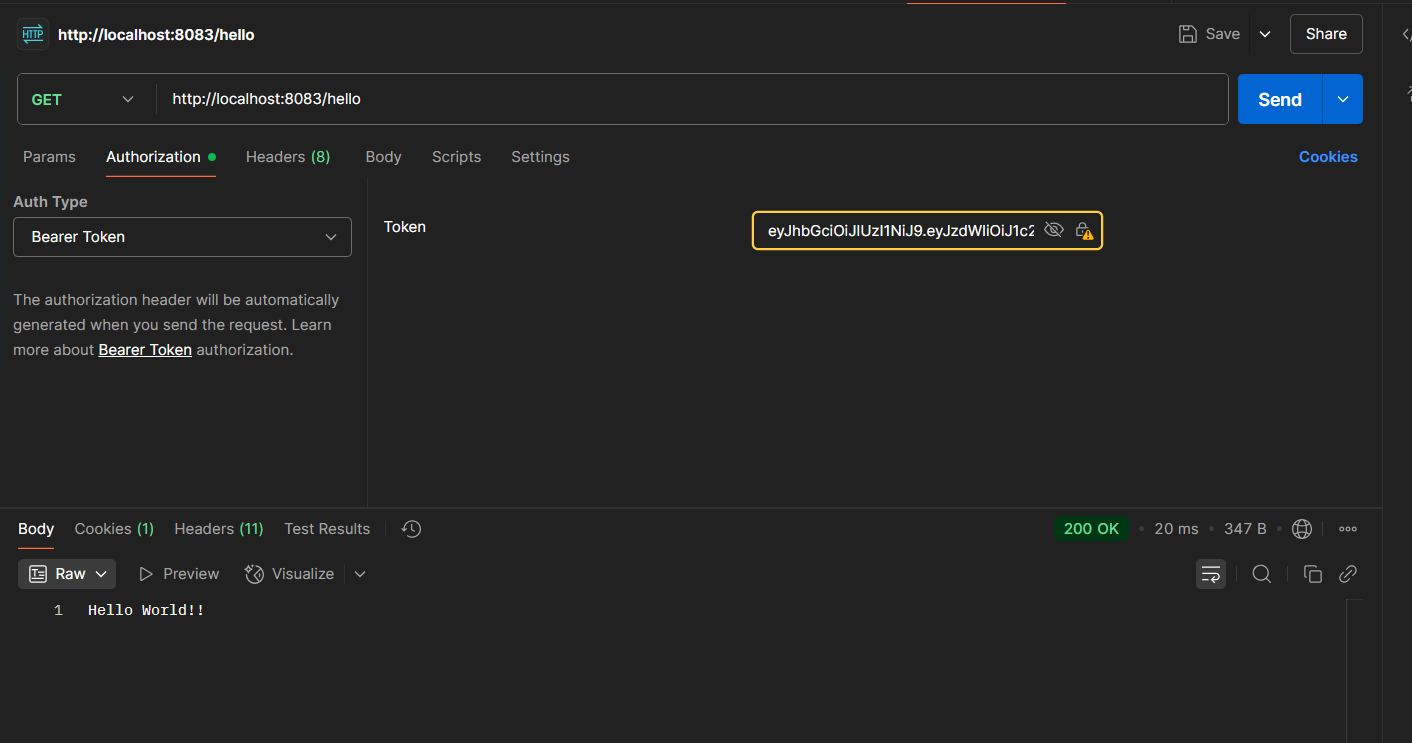


Step 6: Run “SpringLearnApplication.java”

**OUTPUT: Here I have done this in POSTMAN instead of cmd.**

****

**Using that token in endpoint of /hello**



ADDITIONAL HANDS ON

**EXERCISE 1:**

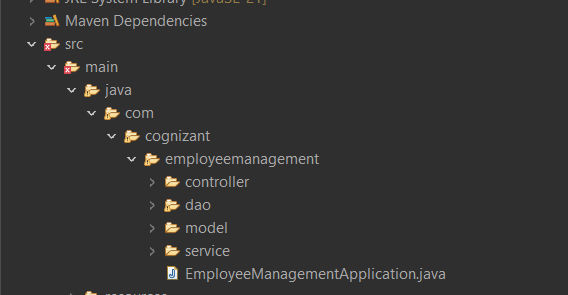
**Problem Statement - Display Employee List and Edit Employee form using RESTful Web Service   
  
In the previous angular module, we developed a screen that lists employees and it was populated with hard coded values. Now this angular application has be changed to get the data from RESTful Web Service 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**

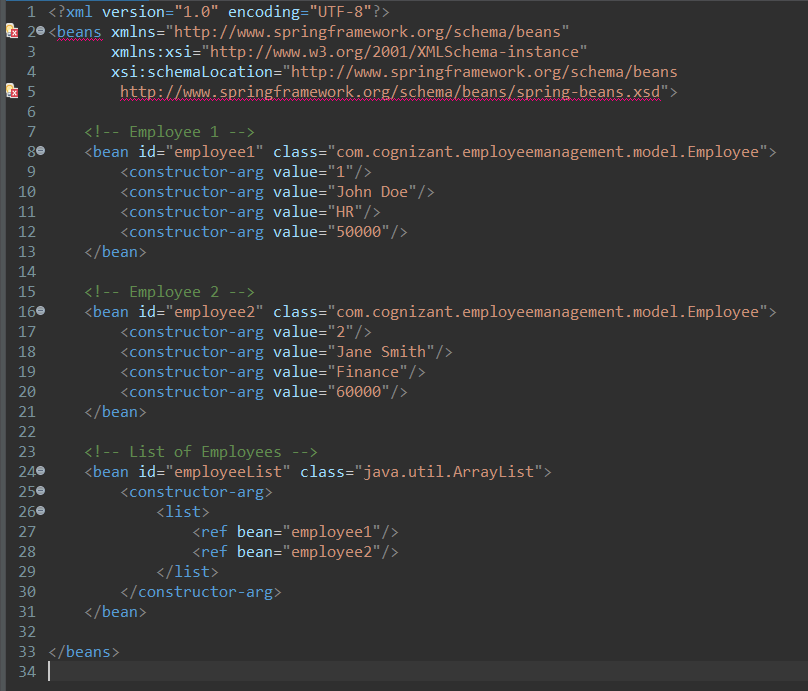
**Once above activities are completed, clicking on the Edit button against each employee should display Edit Employee form with values retrieved from RESTful Web Service. This will also involve activities similar to the one specified above.  
  
NOTE: There is no specific activity as part of this hands on, refer the next hands ons that covers above three activities in detail.**

**SOLUTION:**

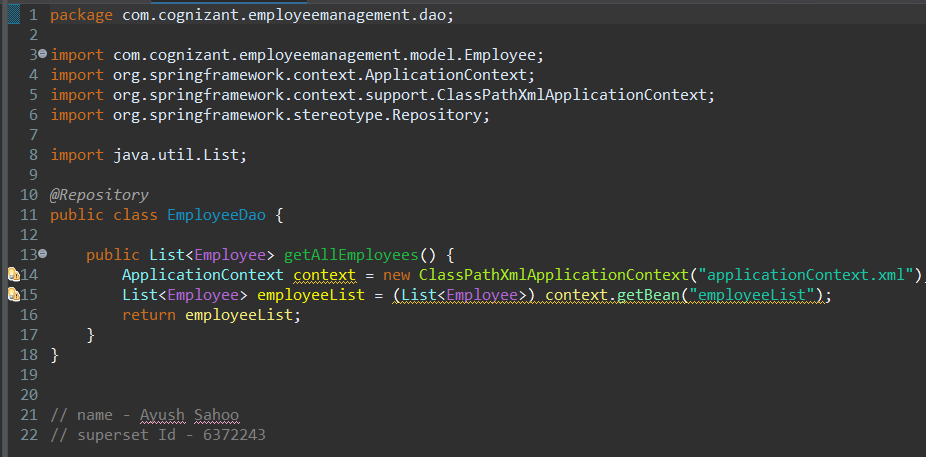
Step 1: Folder Structure



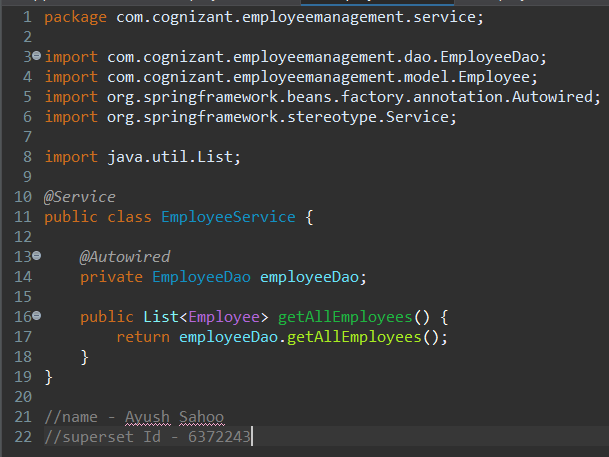
Step 2: Create a “applicationContext.xml” under src/main/resources.



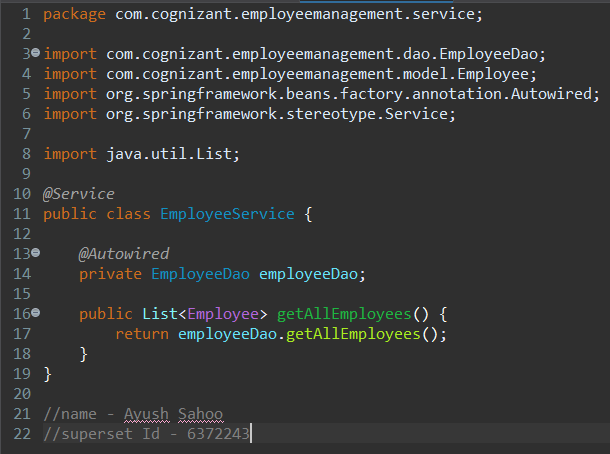
Step 3: Create a “EmployeeDao.java” under src/main/java/com/cognizant/employeemanagement/dao.



Step 4: Create a “EmployeeService.java” under src/main/java/com/cognizant/employeemanagement/service.



Step 5: Create a “EmployeeController.java” under src/main/java/com/cognizant/employeemanagement/controller

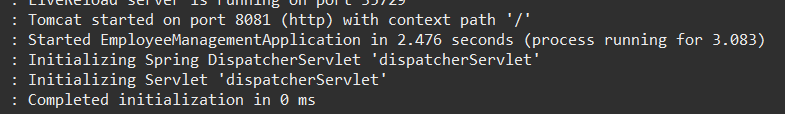


Step 6: Run “Employeemanagement.java”

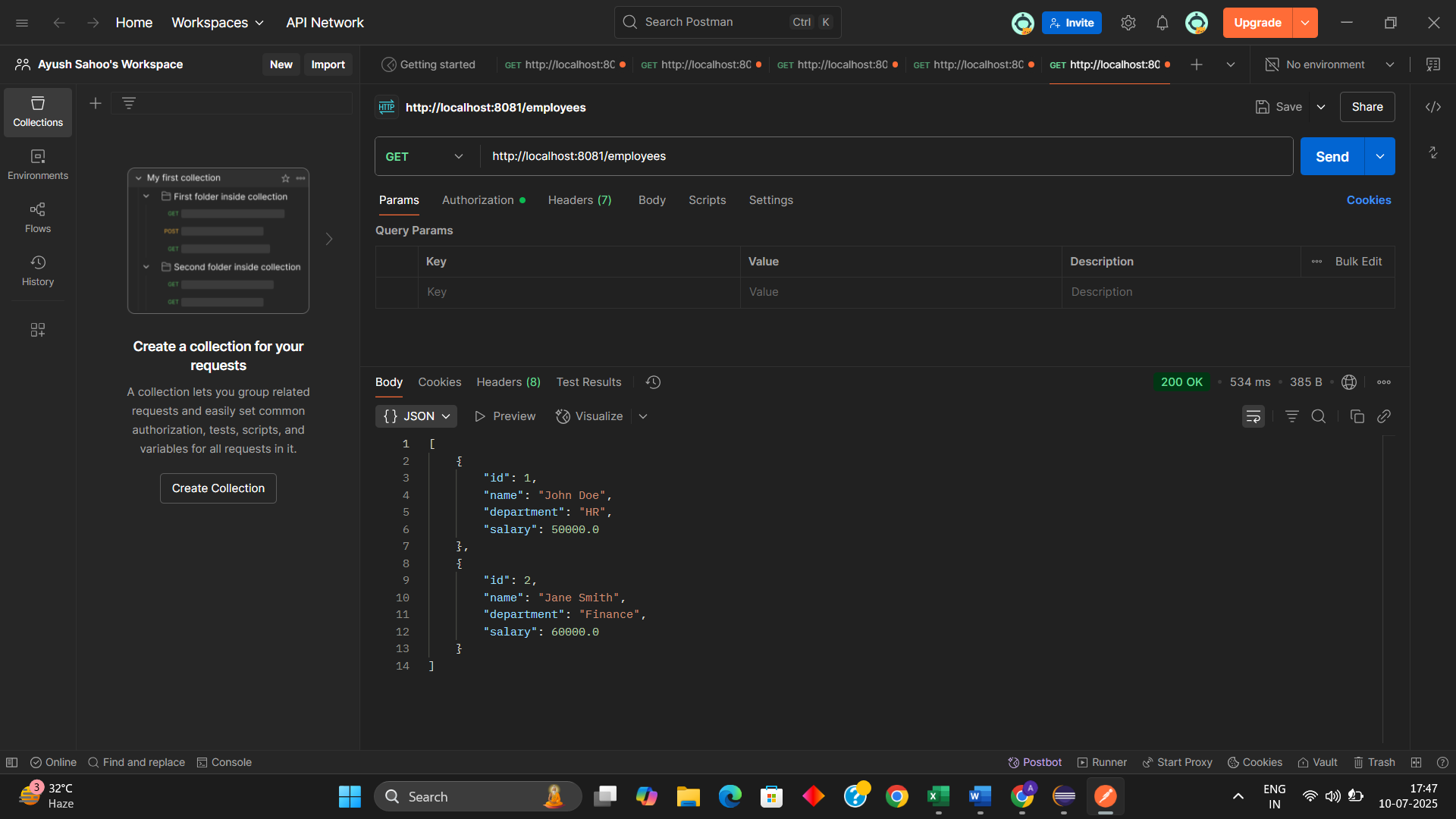


**OUTPUT:**

**Console (Of Eclipse)**

****

**(POSTMAN OUTPUT)**

****

**EXERCISE 2:**

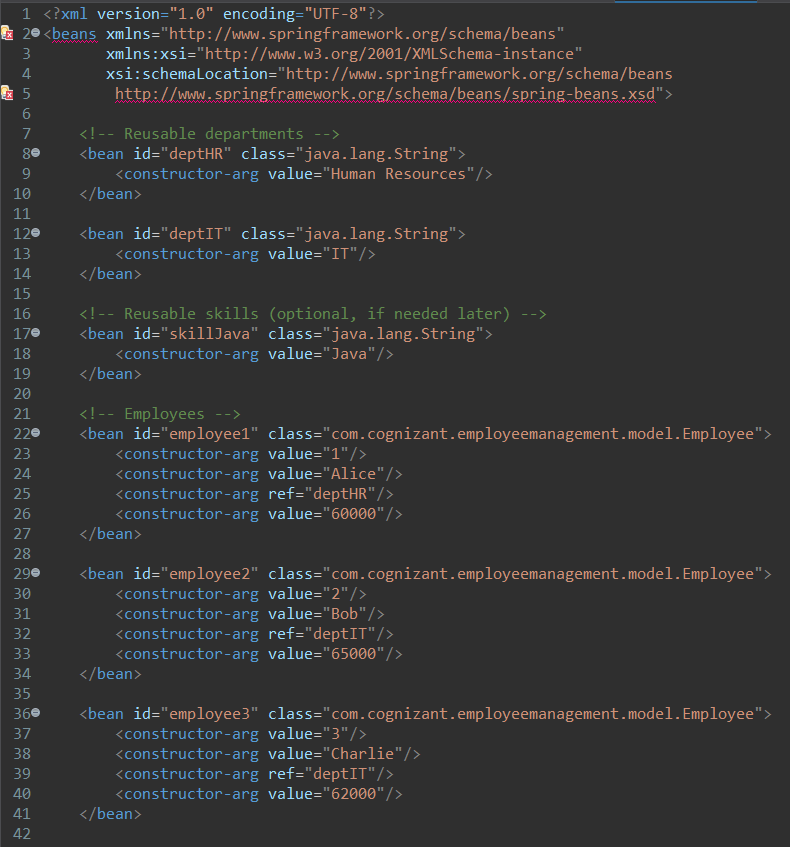
**Create static employee list data using spring xml configuration   
  
Follow steps below to accomplish this activity:**

* **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.**

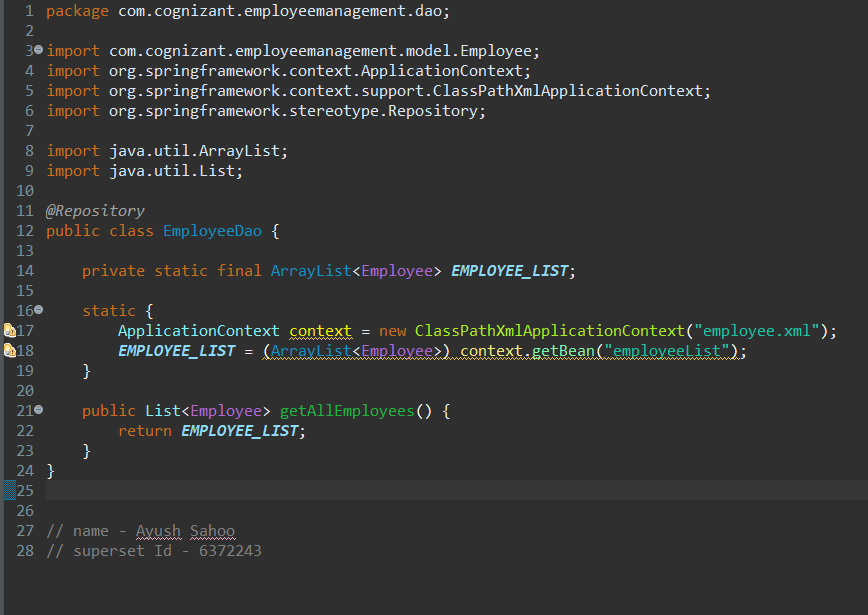
* **In EmployeeDao, incorporate the following:**
  + **Create static variable with name EMPLOYEE\_LIST of type ArrayList<Employee>**
  + **Include constructor that reads employee list from xml config and set the EMPLOYEE\_LIST**
  + **Create method getAllEmployees() that returns the EMPLOYEE\_LIST**

**SOLUTION:**

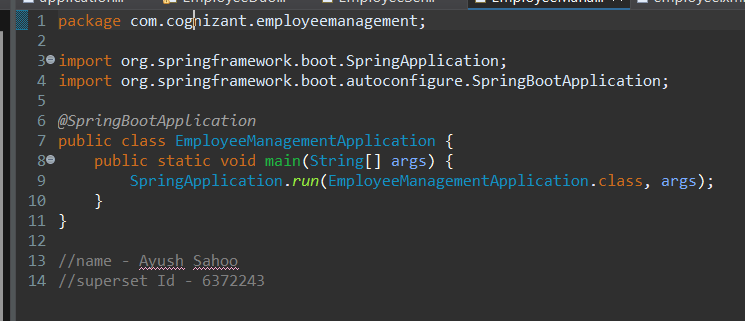
Step 1: Create “Employee.xml” under src/main/resources.



Step 2: Update “EmployeeDao.java”

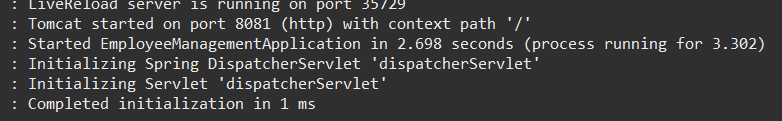


Step 3: Run “EmployeeManagementApplication.java”.

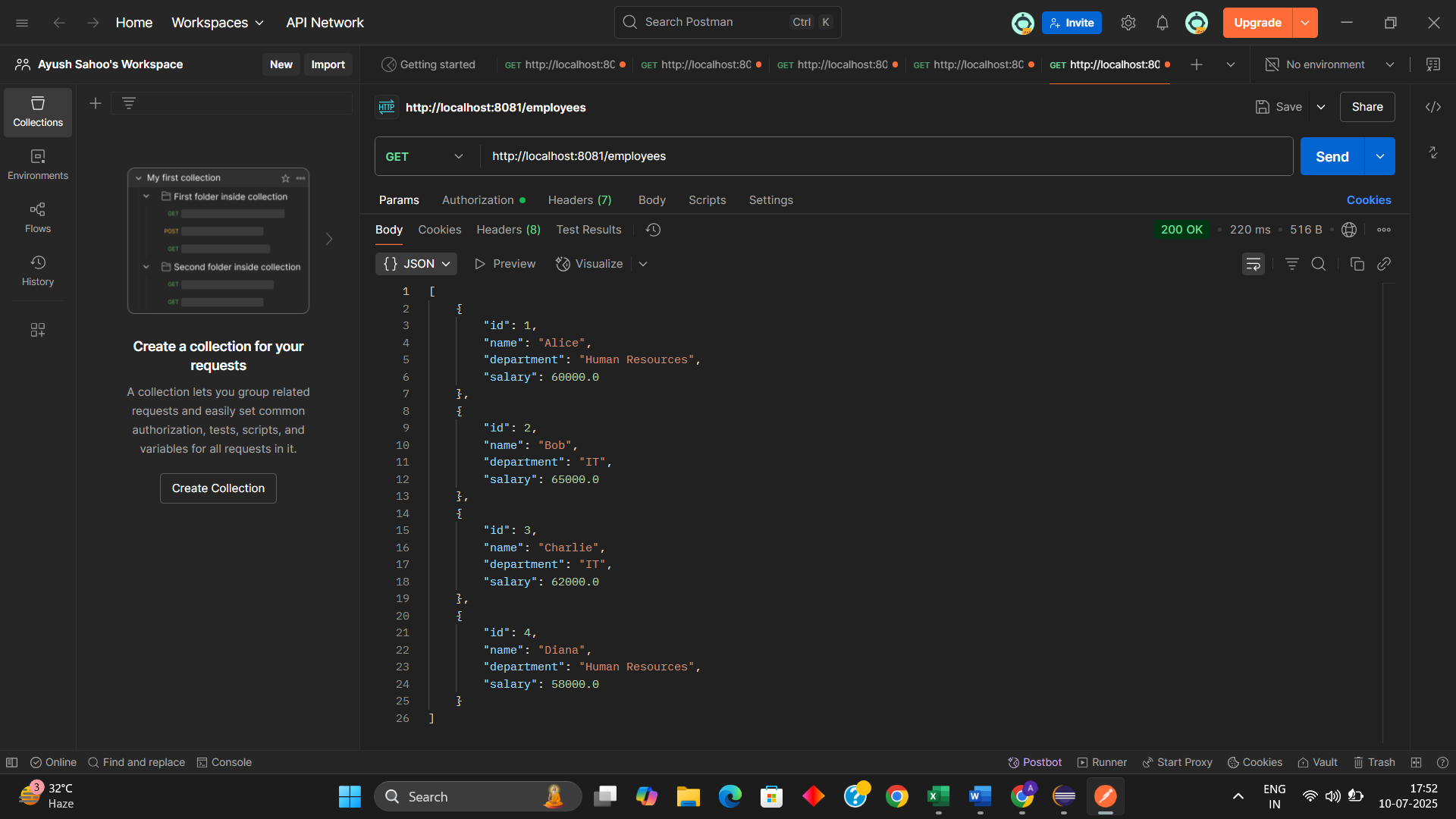


**OUTPUT:**

**(TERMINAL)**

****

**(POSTMAN)**

****

**EXERCISE 3:**

**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.**

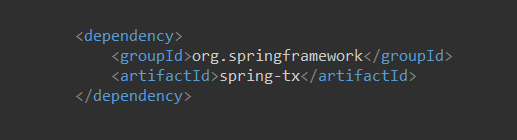
* **In EmployeeController, incorporate the following:**
  + **Include a new get method with name getAllEmployees() that returns the employee list**
  + **Mark this method as GetMapping annotation with the URL as '/employees'**
  + **Within this method invoke employeeService.getAllEmployees() and return the same.**

**​​​​​​**

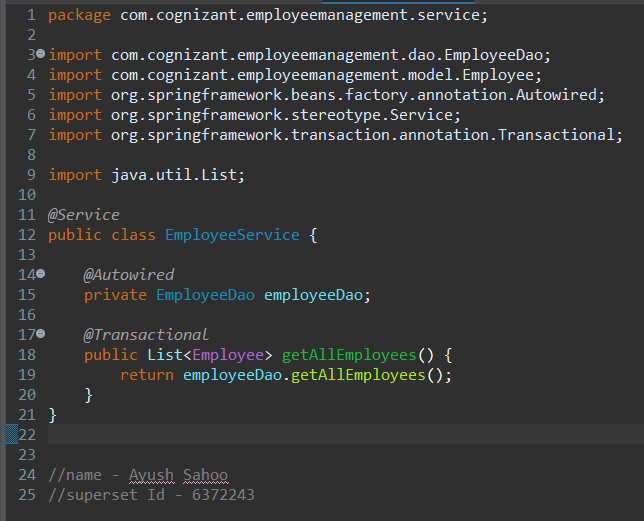
* **Test ​the service using postman.**

**SOLUTION:**

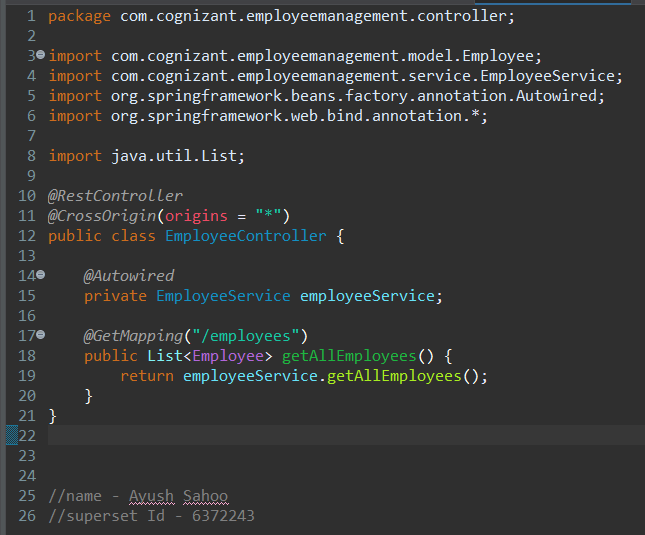
Step 1: Update “pom.xml” with this dependencies.



Step 2: Update “EmployeeService.java”



Step 3: Update “EmployeeController.java”.



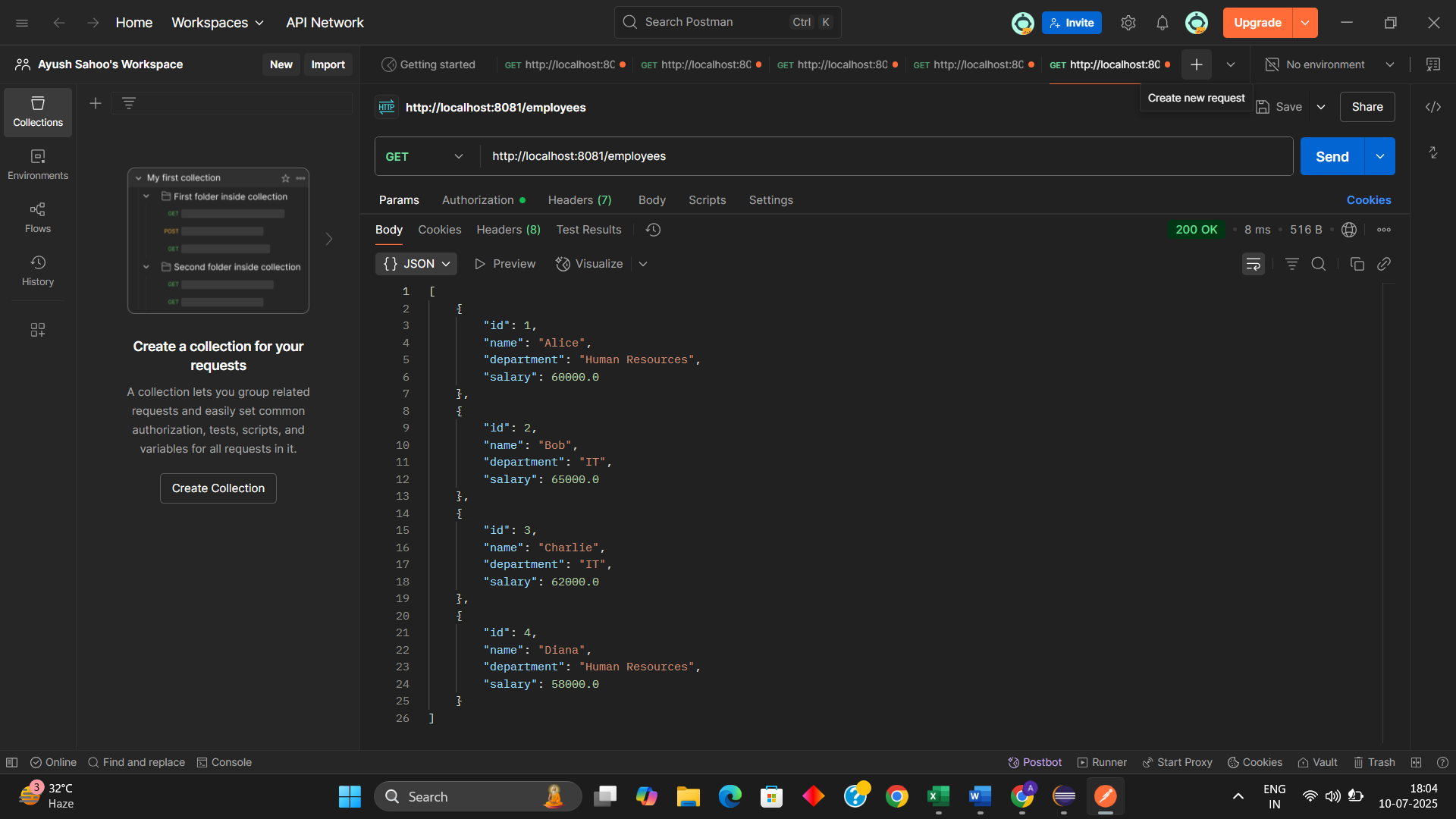
Step 3: Run “EmployeeManagementApplication.java”.

**OUTPUT:**

**(TERMINAL)**

****

**(POSTMAN)**

****

**EXERCISE $:**

**Create REST service for department**

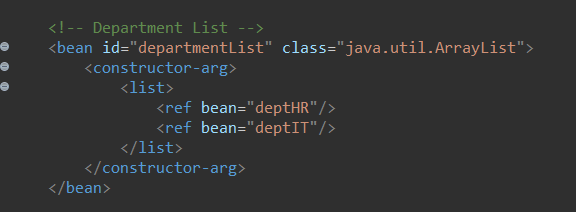
**Create a new service to get all the departments.**

**Follow steps below to achieve this:**

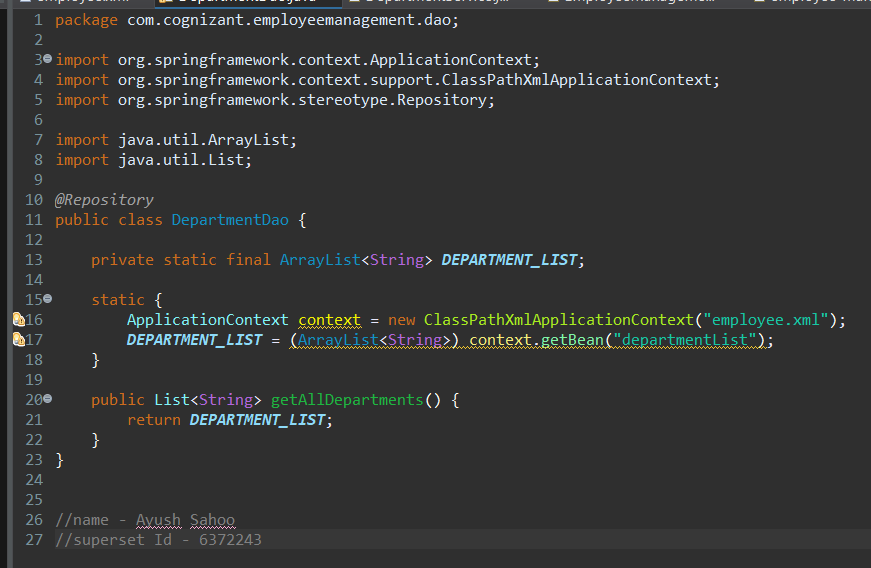
* **Create a new REST Service, define below list of classes and respective methods:**
  + **DepartmentController**
    - **getAllDepartments() with URL "/departments", this method will return array of departments**
  + **DepartmentService**
    - **getAllDepartments()**
  + **DepartmentDao**
    - **getAllDepartments() - 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.**

**SOLUTION:**

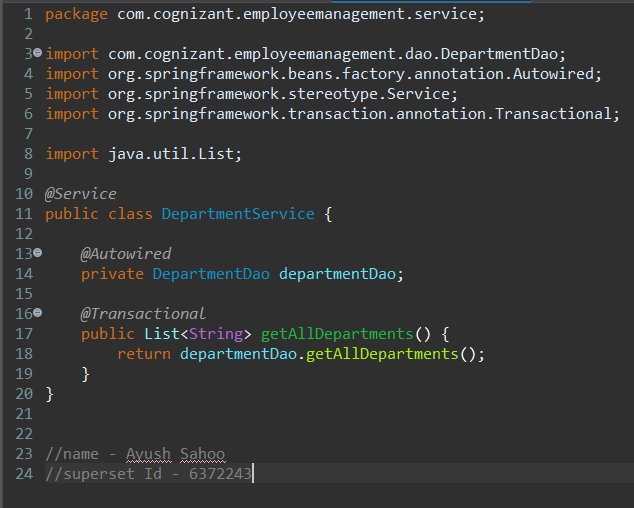
Step 1: Add dependencies in “employee.xml”.



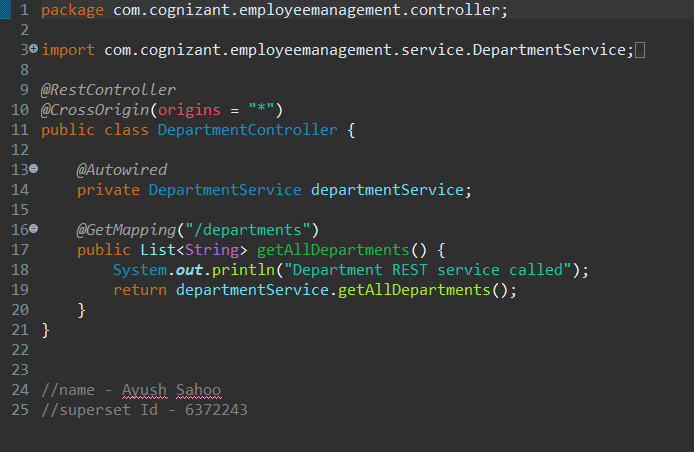
Step 2: Create “DepartmentDao.java” under src/main/java/com/cognizant/employeemanagement/dao.



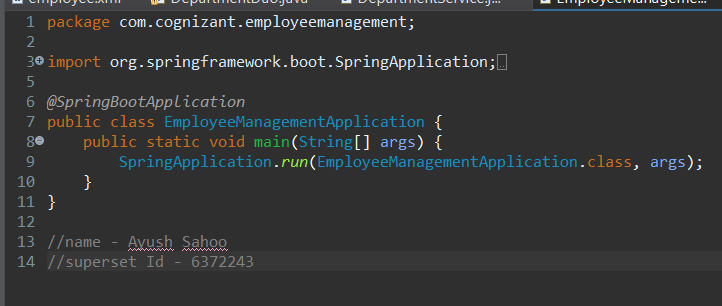
Step 3: Create “DepartmentService.java” under src/main/java/com/cognizant/employeemanagement/service.



Step 4: Create “DepartmentController.java” under src/main/java/com/cognizant/employeemanagement/controller.



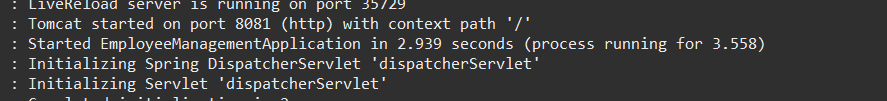
Step 5: Run “EmployeeManagementApplication.java”



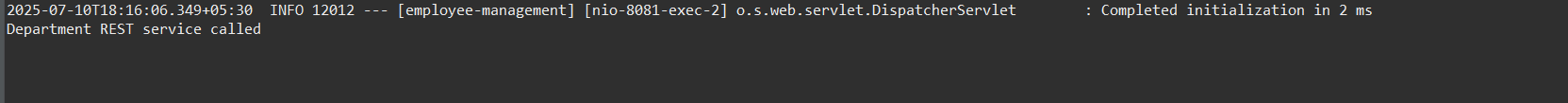
**OUTPUT:**

**(TERMINAL)**

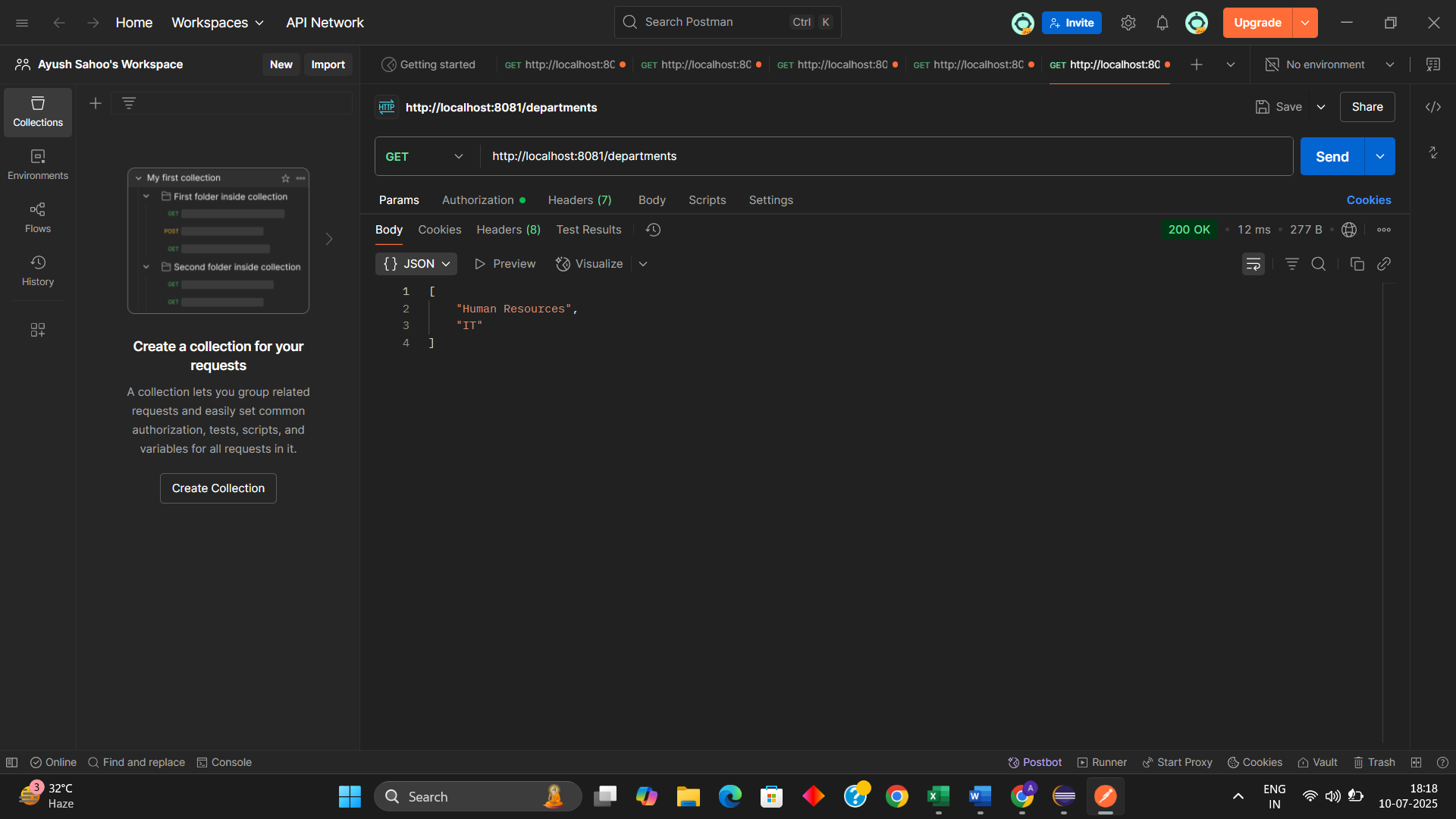
**Before postman**

****

**After postman (RESPONSE LOGS)**

****

**(POSTMAN)**

****

**x----x----x**