Name : Kakumanu Taraka Lakshmi Prasanna

Superset ID : 6314755

Email : 22bq1a4268@vvit.net

Week – 4

Topic : Spring REST using Spring Boot

Hands on 1

Create a Spring Web Project using Maven

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.

I uploaded the project files with the name springRest1-1

5. Purpose of @SpringBootApplication annotation

It is a combination of:

- @Configuration – Marks the class as a source of bean definitions.
- @EnableAutoConfiguration – Enables Spring Boot's auto-configuration.
- @ComponentScan – Scans the package for components and configurations.

Hands on 2

Spring Core – Load Country from Spring Configuration XML

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

bean Tag

Defines a Spring bean in XML. A bean is an object managed by the Spring container.

id Attribute

A unique identifier for the bean used to retrieve it from the container (e.g., "country").

class Attribute

Fully qualified name of the Java class that this bean represents (e.g., com.cognizant.springlearn.Country).

property Tag

Used to inject values into the bean's properties via its setter methods.

name Attribute

Specifies the name of the property in the Java class.

value Attribute

Specifies the literal value to inject into the bean’s property.

• ApplicationContext, ClassPathXmlApplicationContext

ApplicationContext: Spring’s central interface for providing configuration information and managing beans.

ClassPathXmlApplicationContext: Implementation of ApplicationContext that loads context definitions from an XML file located in the classpath.

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

1. Spring locates the bean definition by the given id ("country").
2. It checks if the bean has already been created (singleton scope by default).
3. If not created, it:
    - Instantiates the class (Country)
    - Sets properties via setters
4. Returns the fully initialized bean instance.
5. Logs confirm constructor and methods were invoked.

Hands on 3

Hello World RESTful Web Service

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

Testing in Chrome – Network Tab

- Open **Chrome DevTools** (F12) → **Network Tab**
- Visit http://localhost:8083/hello
- Click on the request → View the **Headers** section

Sample Request Headers:

- Method: GET
- Host: localhost:8083
- User-Agent: Chrome browser

- Accept: text/html

Sample Response Headers:

- Content-Type: text/plain
- Content-Length: 13
- Date: [current date]
- Status: 200 OK

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

1. Open Postman
2. Set:
    - Method: GET
    - URL: http://localhost:8083/hello
3. Click **Send**
4. Response: Hello World!!

Postman → Headers Tab

Response Headers include:

- Content-Type: text/plain;charset=UTF-8
- Transfer-Encoding: chunked
- Date: [server date]

Hands on 4

REST - Country Web Service

• What happens in the controller method?

@RequestMapping("/country") maps HTTP GET requests to the getCountryIndia() method.

Inside the method:

- The Spring application context is initialized from country.xml.
- The Country bean with ID country is fetched.
- This bean is returned as the response.

• How the bean is converted into JSON reponse?

Spring Boot uses **Jackson** (a JSON serialization library) behind the scenes.

When a Java object (like Country) is returned from a method annotated with @RestController, Spring:

- Automatically uses Jackson to serialize the object into JSON.
- Sets the Content-Type of the response to application/json.

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

Network Tab – Chrome DevTools

1. Open Chrome and access http://localhost:8083/country
2. Open DevTools (F12) → Network tab → Click the /country request

Sample Request Headers:

- Method: GET
- Accept: application/json

Sample Response Headers:

- Content-Type: application/json
- Content-Length: 35
- Status: 200 OK
- Date: [current date]

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

Postman – Headers Tab

1. Open Postman and enter http://localhost:8083/country
2. Method: GET → Click **Send**
3. Click **Headers** tab in the response section

Sample Response Headers:

- Content-Type: application/json
- Content-Length: 35
- Server: Apache Tomcat
- Date: [server time]

Hands on 4

REST - Get country based on country code

What Happens in the Controller?

- The @GetMapping("/countries/{code}") annotation maps the URI to the getCountry() method.
- @PathVariable extracts the code from the URI.
- The controller calls the service method to perform business logic.

Service Logic

- Loads the country list bean from country.xml.
- Uses Java Stream and equalsIgnoreCase() to match the code.
- Returns the matched Country object.

JSON Serialization

- Spring Boot automatically converts the Country Java object to JSON using the Jackson library.
- The response has a Content-Type: application/json

Network Tab – Chrome DevTools

Open http://localhost:8083/countries/in, inspect request in **Network tab**.

Request Headers:

- Accept: application/json
- Method: GET

Response Headers:

- Content-Type: application/json
- Status: 200 OK

Postman – Headers Tab

Test the API in Postman using:

bash

CopyEdit

GET http://localhost:8083/countries/in

Response Headers in Postman:

- Content-Type: application/json

- Content-Length: 35
- Server: Apache Tomcat
- Status: 200 OK

Hands on 5

Create authentication service that returns JWT

What happens in the controller?

- Reads Authorization header.
- Decodes credentials.
- Authenticates user.
- Generates and returns JWT as a JSON response.

How is JWT generated?

- JwtUtil creates token using Jwts.builder(), sets subject, issue and expiry date, and signs it using HS256 and a secret key.

Headers in Chrome DevTools

- Use browser or Postman → check **Network tab > Request headers**.
    - Authorization: Basic <base64(user:pwd)>

Headers in Postman

- Under **Headers** tab:
    - Authorization: Basic dXNlcjpwd2Q=
    - Content-Type: application/json (response)