**Spring – REST XML Response**

The popularity of [REST API](https://www.geeksforgeeks.org/rest-api-introduction/) is increasing day by day as it fulfills architecting an application in a convenient manner. A REST API is an acronym for ‘Representational state transfer’. It is also called RESTful web services. It is also a ‘Controller’, but with a difference that Rest Controller returns Data, while Controller returns a View of ‘Model-View-Controller’ architecture. REST API can work on all HTTP methods like ( GET, POST, PUT, PATCH, DELETE, etc ). These methods correspond to create, read, update, and delete ([CRUD](https://www.geeksforgeeks.org/crud-operations-in-mysql/)) operations, respectively. It can return many types of data. JSON is considered the standard form for data transferring between web applications.

**Data types that REST API can return are as follows:**

1. JSON (JavaScript Object Notation)
2. XML
3. HTML
4. XLT
5. Python
6. PHP
7. Plain text

***Pre-requisite:****Though JSON is the standard form to transfer data among applications, it comes with some disadvantages which are overcome in XML format data transferring.*

**The advantages of XML are as follows:**

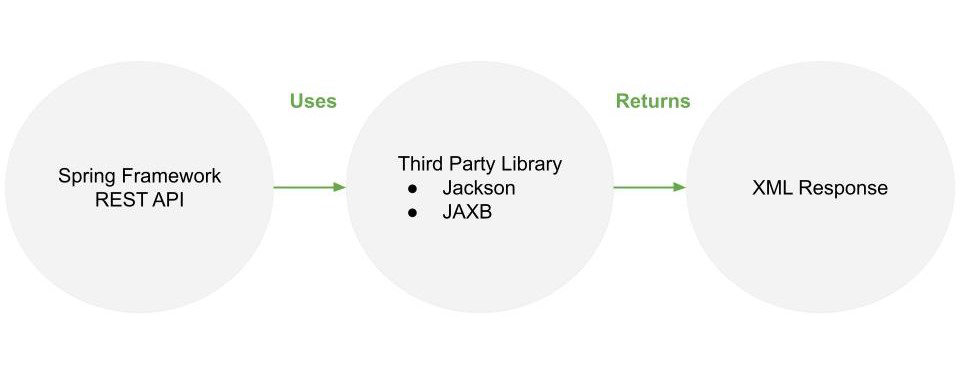
1. It is an Extensible markup language that uses tags for data definition.
2. It supports namespaces.
3. It supports comments.
4. It supports various encoding.
5. Last but not the least, XML is more secure than JSON.

**Note**:

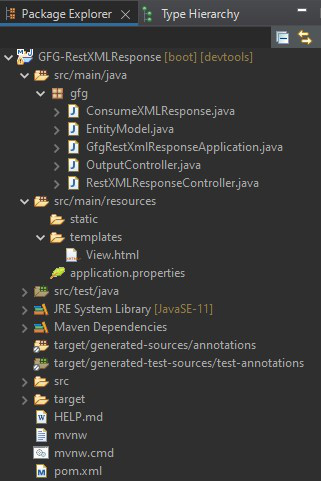
* JSON is less secure because of the absence of a JSON parser in the browser.
* JSONP is dangerous because it allows cross-origin exchanges of data.

**REST API – XML Response**

* When we create a Spring Boot project with ‘Starter Web’ dependency, we only get support for returning data in JSON format, with the help of the Jackson library.
* To embed support for returning data in XML format we need third-party dependencies.
* There are many libraries that support XML return format, for Example – Jackson, JAXB, etc.



*Working of Rest XML Response*



*Project Structure – Maven*

**Way 1: Using Jackson**

The Jackson library is already present in the Spring framework’s classpath for our application. We just need to add an extension to the Jackson library that can work with XML data responses. To add the extension, add the following dependency in the project build.

**Maven – pom.xml**

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

After adding the above dependency and updating the project, the Jackson XML extension will get added in the classpath so the Spring MVC will automatically pick this up for XML responses.

**A. File:**pom.xml (Project configuration)

**Example:**

* XML

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="<http://maven.apache.org/POM/4.0.0>"           xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"           xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <https://maven.apache.org/xsd/maven-4.0.0.xsd>">      <modelVersion>4.0.0</modelVersion>      <parent>          <groupId>org.springframework.boot</groupId>          <artifactId>spring-boot-starter-parent</artifactId>          <version>2.6.3</version>          <relativePath/> <!-- lookup parent from repository -->      </parent>      <groupId>sia</groupId>      <artifactId>GFG-RestXMLResponse</artifactId>      <version>0.0.1-SNAPSHOT</version>      <name>GFG-RestXMLResponse</name>      <description>GeeksforGeeks</description>      <properties>          <java.version>11</java.version>      </properties>      <dependencies>          <dependency>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-starter-thymeleaf</artifactId>          </dependency>          <dependency>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-starter-web</artifactId>          </dependency>            <dependency>              <groupId>com.fasterxml.jackson.dataformat</groupId>              <artifactId>jackson-dataformat-xml</artifactId>          </dependency>            <dependency>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-devtools</artifactId>              <scope>runtime</scope>              <optional>true</optional>          </dependency>          <dependency>              <groupId>org.projectlombok</groupId>              <artifactId>lombok</artifactId>              <optional>true</optional>          </dependency>          <dependency>              <groupId>org.springframework.boot</groupId>              <artifactId>spring-boot-starter-test</artifactId>              <scope>test</scope>          </dependency>      </dependencies>        <build>          <plugins>              <plugin>                  <groupId>org.springframework.boot</groupId>                  <artifactId>spring-boot-maven-plugin</artifactId>                  <configuration>                      <excludes>                          <exclude>                              <groupId>org.projectlombok</groupId>                              <artifactId>lombok</artifactId>                          </exclude>                      </excludes>                  </configuration>              </plugin>          </plugins>      </build>  </project> |

**B.** Bootstrapping of application

**Example:**GfgRestXmlResponseApplication.java

* Java

|  |
| --- |
| // Java Program to Illustrate Bootstrapping of Application   package gfg;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;    @SpringBootApplication  public class GfgRestXmlResponseApplication {      public static void main(String[] args)      {       SpringApplication.run(              GfgRestXmlResponseApplication.class, args);      }  } |

**C.**Object to be return as XML response (EntityModel.java)

This class acts as a User object ( bean ) whose fields will be mapped to XML tags respectively. It also requires Getter/Setter methods which are automatically generated using ‘@Data’ annotation of the ‘Lombok’ library. To embed the Lombok library in the project, add the following dependency in the project build.

Maven – pom.xml

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

**Example:**

* Java

|  |
| --- |
| // Java Program Illustrating Object to be return as XML  // response    package gfg;    // Importing required classes  import lombok.Data;    // Annotation  @Data    // Class  public class EntityModel {        // Class data members      String ID;      String NAME;      String DOB;      String PINCODE;  } |

**D.**REST API returning XML response (RestXMLResponseController.java)

This REST API controller is configured to return data specifically in XML format, using produces attribute of @RequestMapping annotation.

**Example:**

* Java

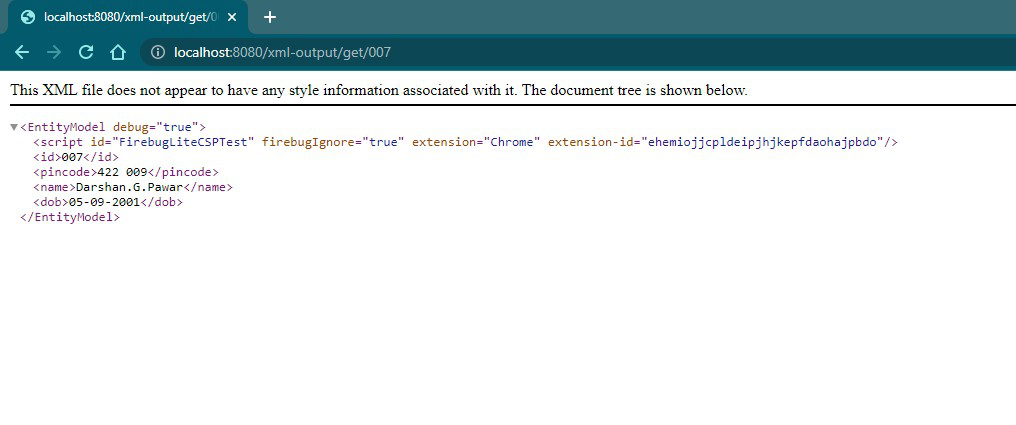
|  |
| --- |
| // Java Program Illustrating REST API returning XML response      // Importing required classes  import org.springframework.http.HttpHeaders;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;    // Annotations  @RestController  @RequestMapping(path = "/xml-output",                  produces = "application/xml")    // Class  public class RestXMLResponseController {        @GetMapping("/get")      public ResponseEntity<EntityModel> get()      {            EntityModel model = new EntityModel();          model.setID("1");          model.setNAME("Darshan.G.Pawar");          model.setDOB("05-09-2001");          model.setPINCODE("422 009");            HttpHeaders hearders = new HttpHeaders();          ResponseEntity<EntityModel> entityModel              = new ResponseEntity<>(model, hearders,                                     HttpStatus.CREATED);            return entityModel;      }        // Annotation      @GetMapping("/get/{id}")        // Class      public ResponseEntity<EntityModel>      getById(@PathVariable("id") String id)      {            EntityModel model = new EntityModel();          model.setID(id);          model.setNAME("Darshan.G.Pawar");          model.setDOB("05-09-2001");          model.setPINCODE("422 009");            HttpHeaders hearders = new HttpHeaders();          ResponseEntity<EntityModel> entityModel              = new ResponseEntity<>(model, hearders,                                     HttpStatus.CREATED);            return entityModel;      }  } |

**Output 1:**



*XML Response returned by get() method*

**Output 2:**



*XML Response returned by getById() method*

**E.** ConsumeXMLResponse.java (Get the REST API response)

* The data returned in XML format by REST API needs to be consumed and make to use.
* Spring Framework provides a convenient way to consume REST responses with the help of Spring’s ‘RestTemplate’.
* It benefits us by preventing the need to write boilerplate code.
* RestTemplate provides us with methods that are HTTP methods specific.
* Here for GET HTTP request, the ‘getForEntity()’ method is used that accepts URL to @GetMapping method, a class to which response will be mapped to and additional object parameters to URL.
* This method will return a ResponseEntity<> object.
* Java

|  |
| --- |
| // Java Program Illustrating Getting REST API Response    // Importing required classes  import org.springframework.http.ResponseEntity;  import org.springframework.web.client.RestTemplate;    // Class  public class ConsumeXMLResponse {        RestTemplate rest = new RestTemplate();        public ResponseEntity<EntityModel> get(String id) {            return rest.getForEntity("<http://localhost:8080/xml-output/get/>{id}",                    EntityModel.class, id);        }  } |

**F.** File: OutputController.java (Regular controller)