Spring - REST JSON Response

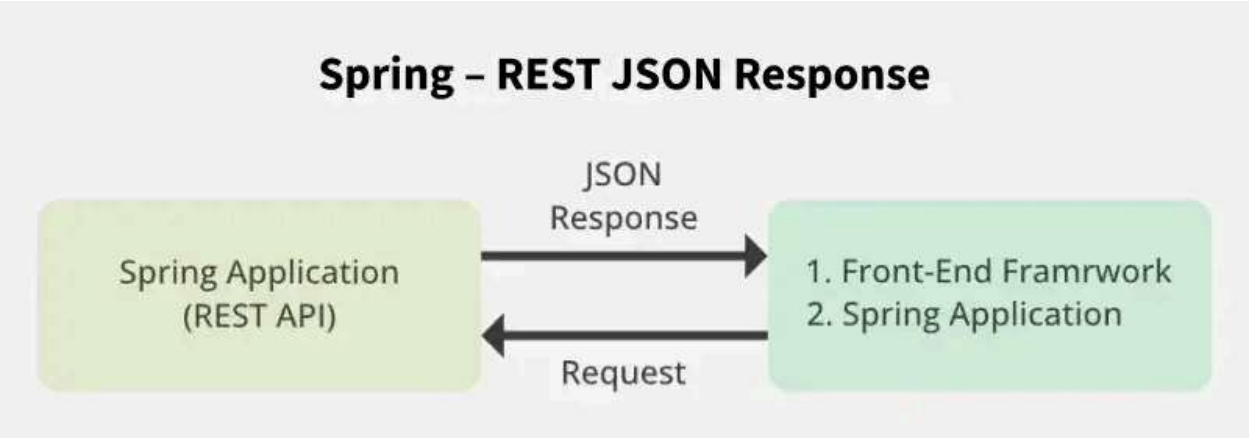
Last Updated : 23 Jul, 2025

REST APIs have become increasingly popular due to their advantages in application development. They operate on a client-server architecture, where the client makes a request, and the server (REST API) responds with data. Clients can be front-end frameworks like Angular, React, or even another Spring application. Data can be sent in various formats, such as plain text, XML, or JSON. Among these, JSON (JavaScript Object Notation) is the standard for transporting data between web applications.

Key benefits of REST APIs:

- **Scalable & Lightweight:** REST APIs are stateless and follow a client-server architecture, which makes them highly scalable.
- **Flexible Data Formats:** They support multiple data formats like JSON and XML, though JSON is the most common format used.
- **Easy Integration:** REST APIs can be easily integrated into front-end frameworks like Angular, React, or even other Spring applications.

The below image demonstrates the communication between a Spring Application (REST API) and a front-end framework, where requests are sent from the front-end and JSON responses are returned.



Prerequisites

The prerequisites required are as follows:

- Basic knowledge of Spring Boot and REST APIs.
- A Spring Boot project set up with Maven or Gradle.
- Familiarity with JSON and its structure.

JSON

JSON is an abbreviation for JavaScript Object Notation. It is a text-based data format following Javascript object syntax. It has syntax somewhat like a Javascript object literal and can be used independently from Javascript. Many programming environments have the ability to parse and generate JSON. It exists as a string and needs to be converted into a native Javascript object to access the data by available global JSON methods of Javascript.

Structure of the JSON

```
{
    "id": 07,
    "framework": "Spring",
    "API": "REST API",
    "Response JSON": true
}
```

When sending an array of objects, they are wrapped in square brackets

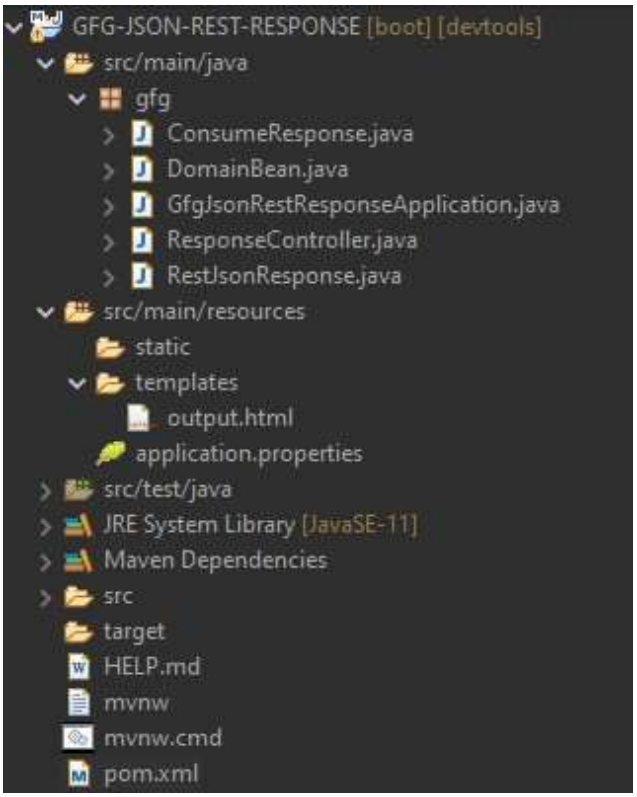
```
[
    {
        "id": 07,
        "name": "darshan"
    },
```

```
    {
        "id": 08,
        "name": "geek"
    }
]
```

Setting Up the Spring Boot Project

Project Structure

The project structure for the application is as follow:



Step 1: Add Dependencies

To create a Spring Boot application that handles JSON responses, we need to include the spring-boot-starter-web dependency. This dependency provides essential features for building RESTful web services.

Maven Dependency:

Add the following dependency to the pom.xml file

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://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-JSON-REST-RESPONSE</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>GFG-JSON-REST-RESPONSE</name>
    <description>REST API Response</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>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
```

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

Step 2: Bootstrapping the Spring Application

Create the main application class to bootstrap the Spring Boot application.

GfgJsonRestResponseApplication.java:

```
// Java Program to Illustrate Bootstrapping of an
// Application

package gfg;

// Importing classes
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

// Annotation
@SpringBootApplication

// Main class
public class GfgJsonRestResponseApplication {

    // Main driver method
```

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

        SpringApplication.run(
            GfgJsonRestResponseApplication.class, args);
    }
}
```

Step 3: Creating the Domain Bean

The domain bean represents the data structure that will be transferred as a JSON response. Use Lombok to automatically generate getters and setters.

Maven Dependency:

Dependency is as depicted below as follows:

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

DomainBean.java:

```
// java Program to illustrate DomainBean class

package gfg;

// Importing class
import lombok.Data;

// Annotation
@Data
// Class
public class DomainBean {

    String id;
    String name;
    String data;
}
```

Step 4: Implementing the REST Controller

The REST controller handles incoming requests and sends JSON responses. Key annotations used include **@RestController**, **@RequestMapping**, and **@GetMapping.**

The below table demonstrates the essential annotation

Annotation	Description
@RestController	Combines @Controller and @ResponseBody
@RequestMapping	Maps web requests with 'path' attribute and response format with 'produces' attribute
@CrossOrigin	Permits cross-origin web requests on the specific handler classes/methods
@GetMapping	Maps GET request on specific handler method
@PathVariable	Binds the URI template variables to method parameters

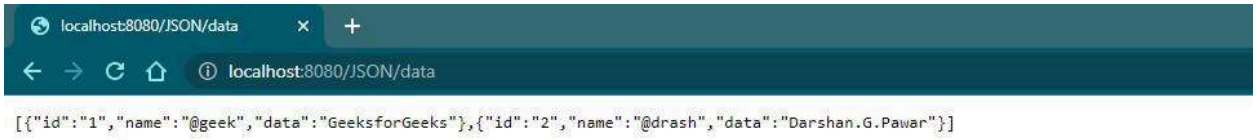
RestJsonResponse.java:

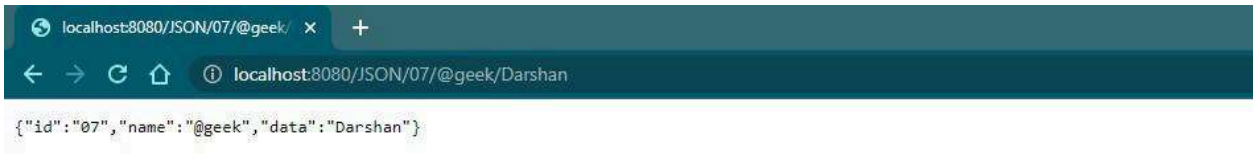
This class provides RESTful services.

```
package gfg;

import java.util.ArrayList;

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

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

@RestController
@RequestMapping(path="/JSON", produces="application/json")
@CrossOrigin(origins="*")
public class RestJsonResponse {

    @GetMapping("/data")
    public ArrayList<DomainBean> get() {

        ArrayList<DomainBean> arr = new ArrayList<>();

        DomainBean userOne = new DomainBean();
        userOne.setId("1");
        userOne.setName("@geek");
        userOne.setData("GeeksforGeeks");

        DomainBean userTwo = new DomainBean();
        userTwo.setId("2");
        userTwo.setName("@drash");
        userTwo.setData("Darshan.G.Pawar");

        arr.add(userOne);
        arr.add(userTwo);

        return arr;
    }

    @GetMapping("/{id}/{name}/{data}")
    public ResponseEntity<DomainBean> getData(@PathVariable("id") String id,
                            @PathVariable("name") String name,
                               @PathVariable("data") String data) {

        DomainBean user = new DomainBean();
        user.setId(id);
        user.setName(name);
        user.setData(data);

        HttpHeaders headers = new HttpHeaders();

        ResponseEntity<DomainBean> entity = new ResponseEntity<>
(user,headers,HttpStatus.CREATED);

        return entity;
    }
}
```

Outputs:





REST API's JSON response can be consumed by:

- Spring application itself.
- Front-end application/framework

Spring Application

- Spring offers the RestTemplate (a convenient way of handling a REST response)
- It has the HTTP method-specific handler methods.

ConsumeResponse.java (Consume REST API response):

```
// Java Program to Illustrate Consume REST API response
package gfg;

// Importing required classes
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

// Class
public class ConsumeResponse {

    // Creating an object of ResponseEntity class
    RestTemplate rest = new RestTemplate();

    public ResponseEntity<DomainBean> get()
```

```
    {

        return rest.getForEntity(
            "http://localhost:8080/JSON/{id}/{name}/{data}",
            DomainBean.class, "007", "geek@drash",
            "Darshan.G.Pawar");
    }
}
```

A normal Spring controller is used to retrieve the responses from the RestTemplate method and returns a view.

ResponseController.java (Regular Controller):

```
// Java Program to Illustrate Regular Controller

package gfg;

// Importing required classes
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

// Annotation
@Controller
@RequestMapping("/ConsumeResponse")

// Class
public class ResponseController {

    @GetMapping("/get") public String get(Model model)
    {

        // Creating object of ConsumeResponse class
        ConsumeResponse data = new ConsumeResponse();
        model.addAttribute("response",
                           data.get().getBody());
        model.addAttribute("headers",
                           data.get().getHeaders());

        return "output";
    }
}
```

Output.html (output of API: Thymeleaf template):

```
<!DOCTYPE html>
<html xmlns="https://www.w3.org/1999/xhtml/"
      xmlns:th="https://www.thymeleaf.org/">
<head>
<title>GeeksforGeeks</title>
</head>
<body>

<h1 style="color:forestgreen" th:text="${response.id}">attributeValue will be
placed here</h1>
<h1 style="color:forestgreen" th:text="${response.name}">attributeValue will
be placed here</h1>
<h1 style="color:forestgreen" th:text="${response.data}">attributeValue will
be placed here</h1>

<h2 style="color:forestgreen; width : 350px"
th:text="${headers}">attributeValue will be placed here</h2>

</body>
</html>
```

Output:

Front-end Application/Framework - Angular

consume-json.component.ts (Angular component):

- This component retrieves the JSON data from the specified URL targeting REST API.
- Retrieved data is stored in a variable.

import { Component, OnInit, Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

```
@Component({
  selector: 'app-consume-json',
  templateUrl: './consume-json.component.html',
  styleUrls: ['./consume-json.component.css']
})
export class ConsumeJsonComponent implements OnInit {
  restData : any;
  constructor(private http:HttpClient) { }
   ngOnInit() {
    this.http.get('http://localhost:8080/JSON/data')
     .subscribe(data => this.restData = data);
   }
}
```

consume-json.component.html (view of component):

- Get the stored data by dot(.) notation.
- 'ngFor' is an Angular template directive used for iterating through the collection of objects.

```
<h1>Rest API Response : JSON</h1>

<div>
 <table *ngFor="let json of restData">
    <tr>
      <td>{{json.id}}</td>
      <td>{{json.name}}</td>
      <td>{{json.data}}</td>
    </tr>
 </table>
</div>
```

consume-json.component.css (Style file):

```
h1, tr{
  color : green;
```

```
    font-size : 25px;
}
```

Add - '<app-consume-json></app-consume-json>' in the 'app.component.html' file

Output:

- You can additionally map an object to JSON object literal using the Jackson API.
- Add the following dependency:

<dependency>

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

<artifactId>jackson-databind</artifactId>

<version>2.5.3</version>

</dependency>

Note: This dependency will also automatically add the following libraries to the classpath:

- jackson-annotations
- jackson-core