Spring Boot - Introduction to RESTful Web Services

Last Updated : 12 Mar, 2025

RESTful Web Services REST stands for **REpresentational State Transfer**. It was developed by **Roy Thomas Fielding**, one of the principal authors of the web protocol HTTP. Consequently, REST was an **architectural approach** designed to make the optimum use of the **HTTP protocol**. It uses the concepts and verbs already present in HTTP to develop web services. This made REST incredibly easy to use and consume, so much so that it is the go-to standard for building web services today.

A resource can be anything, it can be accessed through a **URI (Uniform Resource Identifier)**. Unlike SOAP, REST does not have a standard messaging format. We can build REST web services using many representations, including both XML and JSON, although JSON is the more popular option. An important thing to consider is that REST is not a standard but a style whose purpose is to constrain our architecture to a client-server architecture and is designed to use stateless communication protocols like HTTP.

Important Methods of HTTP

The main methods of HTTP we build web services for are:

1. **GET**: Reads existing data.
2. **PUT**: Updates existing data.
3. **POST**: Creates new data.
4. **DELETE**: Deletes the data.

1. GET

The default request method for HTTP. We don't have any request body with this method, but we can define multiple request parameters or path variables in the URL. This method is used for obtaining some resources. Depending on the presence of an ID parameter, we can either fetch a specific resource or fetch a collection of resources in the absence of the parameter. Sample GET request in Spring Boot Controller:

```
@GetMapping("/user/{userId}")
public ResponseEntity<Object> getUser(@PathVariable int userId) {
    UserEntity user = userService.getUser(userId);
    return new ResponseEntity<>(user, HttpStatus.OK);
}
```

Example of GET operation to perform in an application:

- **GET/employees**: This will retrieve all employee details.

2. POST

The POST method of HTTP is used to create a resource. We have a request body in this method and can also define multiple request parameters or path variables in the URL. Sample POST request in Spring Boot Controller:

```
@PostMapping(value = "/user")
public ResponseEntity<Object> addUser(@RequestBody UserEntity
user) {
    userService.saveOrUpdate(user);
    return new ResponseEntity<>("User is created successfully",
HttpStatus.CREATED);
}
```

Example of POST operation to perform in an application:

- **POST/employees**: This will create an employee.

3. PUT

The PUT method of HTTP is used to update an existing resource. We have a request body in this method and can also define multiple request parameters or path variables in the URL. Sample PUT request in Spring Boot Controller:

```
@PutMapping("/user/{userId}")
public ResponseEntity<Object> updateUser(@PathVariable int
userId, @RequestBody UserEntity user) {
    userService.saveOrUpdate(user);
    return new ResponseEntity<>("User is updated successfully",
HttpStatus.OK);
}
```

Example of PUT operation to perform in an application:

- **PUT/employees/{id}**: This will update an existing employee's details.

4. DELETE

The DELETE method of HTTP is used to remove a resource. We don't have a request body in this method but can define multiple request parameters or path variables in the URL. We can delete multiple or single records, usually based on whether we have an ID parameter or not. We can delete multiple or single records, usually based on whether we have an ID parameter or not. Sample DELETE request in Spring Boot Controller:

```
@DeleteMapping("/user/{userId}")
public ResponseEntity<Object> deleteUser(@PathVariable int
userId) {
    userService.deleteUser(userId);
    return new ResponseEntity<>("User is deleted successfully",
HttpStatus.OK);
}
```

Example of DELETE operation to perform in an application:

- **DELETE/employees**: This will delete all employees.

REST web services use the Status-Line part of an HTTP response message to inform clients of their request's ultimate result.

HTTP Standard Status Codes

The status codes defined in HTTP are the following:

- **200**: Success
- **201**: Created
- **401**: Unauthorized
- **404**: Resource Not Found
- **500**: Server Error

Uses of Spring Boot - REST

The web services are completely stateless. The service producer and consumer have a mutual understanding of the context and content being passed along. When there is already some catching infrastructure present since we can use those to enhance performance in a REST API. This is so since idempotent requests like GET requests are considered cacheable, modern caching mechanisms allow POST requests to be cached under certain conditions when using **Cache-Control headers**. Often Bandwith is of significant importance to organizations. Rest is instrumental then as there are no overhead headers from the SOAP XML payload. Web service delivery or aggregation into existing websites can be enabled easily with a RESTful style. It's simply not required to overhaul the architecture since we can expose the API as an XML and consume the HTML web pages, thus still maintaining the external contract of the service.

Principles of RESTful Web Services

The following are the main principles rest services follow, which makes them fast, lightweight, and secure are:

- Resource Identification through URI- A RESTful web service provides an independent URI/ global ID for every resource.

- Uniform Interface- Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.
- Self-descriptive messages- Resources and representations are decoupled in a RESTful web service. This allows us to represent the payload in various formats such as HTML, XML, plain text, PDF, JPEG, JSON, and others based on our use case.
- Stateful Interaction through hyperlinks- Every interaction with a resource is stateless; that is, request messages are self-contained.

REST API Security

Modern RESTful APIs should follow the below security practices:

- **Authentication and Authorization:** Use OAuth 2.0 or JWT-based authentication.
- **Rate Limiting:** Implement API rate limiting to prevent abuse.
- **Input Validation:** Sanitize input to prevent SQL injection and XSS attacks.
- **HTTPS Enforcement**: Ensure secure communication by enforcing HTTPS.

Advantages of RESTful Web Services

Some of the primary advantages of using RESTful web services are,

- **Easy to Build**: REST APIs are simpler to build than a corresponding SOAP API. Hence, REST is the better choice if we want to develop APIs quickly.
- **Independent**: Since the client and server are decoupled in RESTful web services, it allows for independent development across projects.
- **Scalability**: Stateless communication and a replicated repository provide a high level of scalability. Compared to SOAP, scaling up an existing website is easier with the REST web services.
- **Layered System:** REST web services have their application divided into multiple layers forming a hierarchy. It makes the application both