RESTful Web Services

SKILLO

Table of Contents

REST

- Was first introduced by Roy Fielding in 2000.
- Stands for **REpresentational State Transfer**.
- Uses HTTP Protocol.
- Revolves around the statement that every component is a resource and it is accessed by a common interface using HTTP standard methods
- In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources.

REST

- Following four HTTP methods are commonly used in REST based architecture.
  - **GET** − Provides a read only access to a resource.
  - **POST** − Used to create a new resource.
  - **DELETE** − Used to remove a resource.
  - **PUT** − Used to update a existing resource or create a new resource.

RESTful Web Service

- A web service is a collection of open protocols and standards used for exchanging data between applications or systems.

- Software applications written in various programming languages and running on various platforms can use web services to exchange data. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

- Web services based on REST Architecture are known as RESTful web services. They uses HTTP methods to implement the concept of REST architecture.

- A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

RESTful Resources

- REST architecture treats every content as a resource.
- Resources can be Text Files, Html Pages, Images, Videos, etc.
- REST Server simply provides access to resources and REST client accesses and modifies the resources.
- A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database.
- Each resource is identified by URIs/ Global IDs.
- REST uses various representations to represent a resource. The most popular representations of resources is JSON.

RESTful Resources

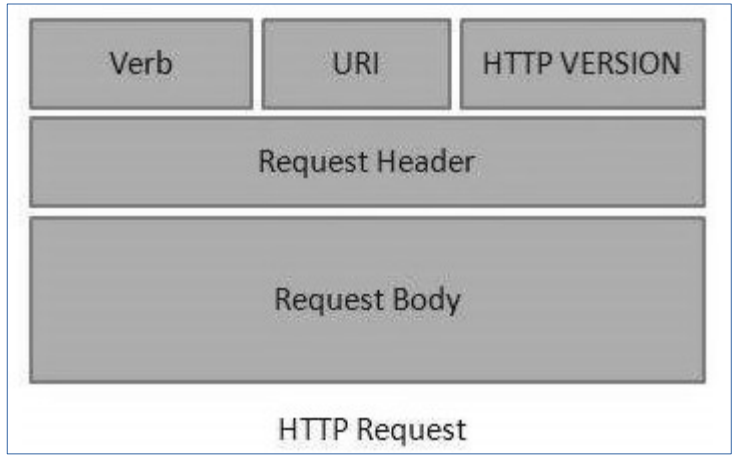
- REST architecture treats every content as a resource.
- Resources can be Text Files, Html Pages, Images, Videos, etc.
- REST Server simply provides access to resources and REST client accesses and modifies the resources.
- A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database.
- Each resource is identified by URIs/ Global IDs.
- REST uses various representations to represent a resource. The most popular representations of resources is JSON.

RESTful Messages

- RESTful Web Services make use of HTTP protocols as a medium of communication between client and server.
- A client sends a message in form of a HTTP Request
- The server responds in the form of an HTTP Response.
- This technique is termed as Messaging.
- These messages contain message data and metadata i.e. information about message itself.

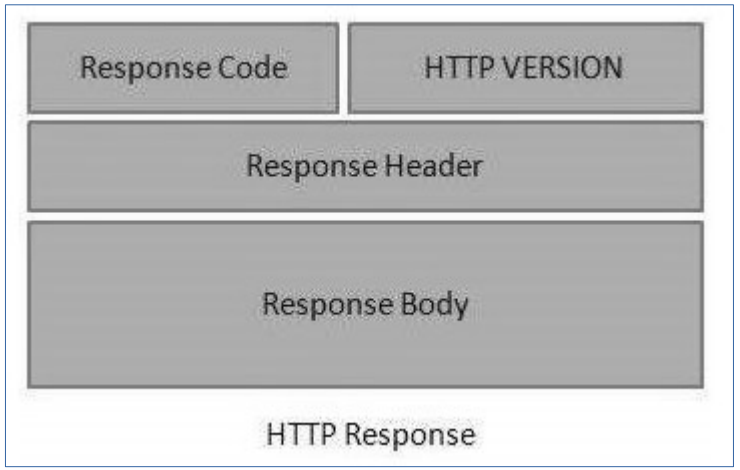
RESTful Messages – HTTP Request

- An HTTP Request has five major parts
  - **Method/Verb** – Indicates the HTTP methods such as **GET, POST, DELETE, PUT**, etc.
  - **URI** – Uniform Resource Identifier (URI) to identify the resource on the server.
  - **HTTP Version** – Indicates the HTTP version. For example, HTTP v1.1.
  - **Headers** – Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by the client, format of the message body, cache settings, etc.
  - **Body** – Message content or Resource representation.



RESTful Messages – HTTP Response

- An HTTP Response has four major parts
  - **Status Code** − Indicates the Server status for the requested resource. For example, 404 means resource not found and 200 means request has been processed successfully.
  - **HTTP Version** − Indicates the HTTP version. For example HTTP v1.1.
  - **Headers** − Contains metadata for the HTTP Response message as keyvalue pairs. For example, content length, content type, response date, server type, etc.
  - **Body** − Response message content or Resource representation.



RESTful Addressing

- Addressing refers to locating a resource or multiple resources lying on the server. It is analogous to locate a postal address of a person.
- Each resource in REST architecture is identified by its URI (Uniform Resource Identifier). A URI is of the following format

```
<protocol>://<service-name>/<ResourceType>/<ResourceID>
```

- Another important attribute of a request is VERB which identifies the operation to be performed on the resource

RESTful Addressing

- The following are important points to be considered while designing a URI.
  - **Use Plural Noun** − Use plural noun to define resources. For example, we've used users to identify users as a resource.
  - **Avoid using spaces** − Use underscore (_) or hyphen (-) when using a long resource name. For example, use authorized_users instead of authorized%20users.
  - **Use lowercase letters** − Although URI is case-insensitive, it is a good practice to keep the url in lower case letters only.
  - **Maintain Backward Compatibility** − As Web Service is a public service, a URI once made public should always be available. In case, URI gets updated, redirect the older URI to a new URI using the HTTP Status code, 300.
  - **Use HTTP Verb** − Always use HTTP Verb like GET, PUT and DELETE to do the operations on the resource. It is not good to use operations name in the URI.

RESTful Methods

- RESTful web service makes heavy uses of HTTP verbs to determine the operation to be carried out on the specified resource(s).
  - **GET** − retrieve a resource from the server. Read-only. Should never modify the state. Cannot have a body.
  - **POST** − create a resource on the server. Non-Idempotent. Multiple invocations can trigger different results.
  - **PUT** − update the state of a resource on the server. Idempotent.
  - **DELETE** − remove a resource from the server. Idempotent.
  - **OPTIONS** − used to request information about the communication options available for the target resource. The response may include an Allow header indicating allowed HTTP methods on the resource, or various Cross Origin Resource Sharing headers.
  - **HEAD** – requests the headers that would be returned if the HEAD request's URL was instead requested with the HTTP GET method. For example, if a URL might produce a large download, a HEAD request could read its Content-Length header to check the filesize without actually downloading the file

RESTful Methods

- Here are important points to be considered:
  - **GET** operations are read only and are safe.
  - **PUT** and **DELETE** operations are idempotent means their result will always be same no matter how many times these operations are invoked
  - **PUT** and **POST** operation are nearly same with the difference lying only in the result where **PUT** operation is idempotent and **POST** operation can cause different result
  - **PUT** is usually used for editing a resource. Also with PUT we already know the identifier of the resource
  - **POST** is usually used for adding a resource to a collection. Often the result can be the new state of that collection. We usually do not know the identifier of the resource when making POST request

RESTful Statelessness

- As per the REST architecture, a RESTful Web Service should not keep a client state on the server
- It is the responsibility of the client to pass its context to the server and then the server can store this context to process the client's further request
- Since the server does not store any information about the client, the endpoints can be hit from a browser, java based client, Postman, Swagger or any other http client and the server will always respond with the same result based on given request parameters.

- Advantages:
  - Web Services can treat each method request independently
  - No need to maintain the client`s previous interactions
- Disadvantages:
  - Web Services need to get extra information in each request and then interpret to get the client`s state.

RESTful Caching

- Caching refers to storing the server response in the client itself, so that a client need not make a server request for the same resource again and again.
- A server response should have information about how caching is to be done, so that a client caches the response for a time-period or never caches the server response.
- Common headers provided by the server that can configure the client`s caching
  - **Date** - Date and Time of the resource when it was created.
  - **Last Modified** - Date and Time of the resource when it was last modified.
  - **Cache-Control** - Primary header to control caching.
  - **Expires** - Expiration date and time of caching.
  - **Age** - Duration in seconds from when resource was fetched from the server.

RESTful Caching

- Best Practices
  - Always keep static contents like images, CSS, JavaScript cacheable, with expiration date of 2 to 3 days.
  - Never keep expiry date too high.
  - Dynamic content should be cached for a few hours only.

RESTful Security

- Best Practices when designing a RESTful Web Service
  - **Validation** − Validate all inputs on the server. Protect your server against SQL or NoSQL injection attacks.
  - **Session Based Authentication** − Use session based authentication to authenticate a user whenever a request is made to a Web Service method.
  - **No Sensitive Data in the URL** − Never use username, password or session token in a URL, these values should be passed to Web Service via the POST (or PUT) method.
  - **Restriction on Method Execution** − Allow restricted use of methods like GET, POST and DELETE methods. The GET method should not be able to delete data.
  - **Validate Malformed XML/JSON** − Check for well-formed input passed to a web service method.
  - **Throw generic Error Messages** − A web service method should use HTTP error messages like 403 to show access forbidden, etc.

RESTful Security

- Common Status codes used in RESTful Services for generic Error Messages
  - **200 - OK** – shows success.
  - **201 - CREATED** – when a resource is successfully created using POST or PUT request. Returns link to the newly created resource using the location header.
  - **204 - NO CONTENT** – when response body is empty. For example, after a DELETE request.
  - **400 - BAD REQUEST** – states that an invalid input is provided. For example, validation error, missing data.
  - **401 - UNAUTHORIZED** – states that user is using invalid or wrong authentication token.
  - **403 - FORBIDDEN** – states that the user is not having access to the method being used. For example, Delete access without admin rights.
  - **404 - NOT FOUND** – states that the endpoint is not available.
  - **409 - CONFLICT** – states conflict situation while executing the method. For example, adding duplicate entry.
  - **500 - INTERNAL SERVER ERROR** – states that the server has thrown some exception while executing the method.

Time for Questions

SKILLO