* What is REST?

REST (REpresentational State Transfer) is an architectural style for developing web services and uses HTTP Protocol. REST is used by many application developer because of the simplicity and the fact that it builds upon existing systems and features of the internet's HTTP (Hypertext Transfer Protocol). In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other. This means that the code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client. It provide interoperability between computer systems on theInternet. In a RESTful web service, requests made to a resource'sURI (Uniform Resource Identifier) will elicit a response with apayload formatted in eitherHTML,XML,JSON, or some other format. The response can confirm that some alteration has been made to the stored resource, and the response can providehypertext links to other related resources or collections of resources. WhenHTTP is used, as is most common, the operations available are: GET, POST, PUT, DELETE.

* How REST works?



Fig. RESTful web service works

* HTTP Methods:
  + GET: The GET method requests a representation of the specified resource. Requests using GET should onlyretrieve data and should have no other effect.

Ex. Note that the query string (name/value pairs) is sent in the URL of a GET request: https://www.xyz.com/test/demo\_form.php?name1=value1&name2=value2

Advantages:

* Since the data sent by the GET method are displayed in the URL, it is possible to bookmark the page with specific query string values
  + - GET requests can be cached
    - GET requests remain in the browser history
    - GET requests can be bookmarked
    - GET requests should never be used when dealing with sensitive data
    - GET requests have length restrictions
    - GET requests is only used to request data (not modify)

Disadvantages:

* + - The GET method is not suitable for passing sensitive information such as the username and password, because these are fully visible in the URL query string as well as potentially stored in the client browser's memory as a visited page.
    - Form data are sent using the URL as the Query string, which is a security risk as anybody can see it.
    - Only 2048 characters can be sent, which is the maximum length of the URL
    - Only ASCII characters are allowed
* You probably should use GET for:
  + - * Search results pages,
      * getting images, scripts, stylesheets,
      * anything which isn't sending data as a part of the request.
  + POST: The PUT method requests that the enclosed entity be stored under the suppliedURI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.

Ex. The data sent to the server with POST is stored in the request body of the HTTP request: POST /test/demo\_form.php HTTP/1.1

Host: w3schools.com

name1=value1&name2=value2

Advantages:

* Form data is sent using the body of the message. Hence, more secure than the GET
* No limitation on length of form data
* No restriction on data type. Binary data are also allowed

Disadvantages:

* Cannot be bookmarked.
* Cannot be cached.
* You probably should use POST for:
* Payment forms (with HTTPS)
* password and login forms (again, with HTTPS)
* posting blog articles, status updates, etc. via a form
  + PUT: The PUT method requests that the enclosed entity be stored under the suppliedURI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI. The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times. The PUT method completely replaces whatever currently exists at the target URL with something else. With this method, you can create a new resource or overwrite an existing one given you know the exact Request-URI.

An example of a PUT method being used to create a new resource would resemble the following: PUT /forums/<new\_thread> HTTP/2.0  
 Host: https://yourwebsite.com/

* + DELETE: The DELETE method deletes the specified resource. The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method may be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server should not indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

A successful response should be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries should be treated as stale. Responses to this method are not cacheable

* HTTP Status Code:

HTTP status codes are standard response codes given by web site servers on the internet. The codes help identify the cause of the problem when a web page or other resource does not load properly.

|  |  |  |
| --- | --- | --- |
| **Status Code** | **Message** | **Description** |
| **Information** | | |
| 100 | Continue | Only a part of the request has been received by the server, but as long as it has not been rejected, the client should continue with the request. |
| 101 | Switching Protocols | The server switches protocol. |
| **Successful** | | |
| 200 | OK | The request is OK. |
| 201 | Created | The request is complete, and a new resource is created . |
| 202 | Accepted | The request is accepted for processing, but the processing is not complete. |
| **Client Error** | | |
| 400 | Bad Request | The server did not understand the request. |
| 401 | Unauthorized | The requested page needs a username and a password. |
| 403 | Forbidden | Access is forbidden to the requested page. |
| 404 | Not Found | The server can not find the requested page. |
| **Server Error** | | |
| 500 | Internal Server Error | The request was not completed. The server met an unexpected condition. |
| 501 | Not Implemented | The request was not completed. The server did not support the functionality required. |
| 502 | Bad Gateway | The request was not completed. The server received an invalid response from the upstream server. |
| 503 | Service Unavailable | The request was not completed. The server is temporarily overloading or down. |
| 504 | Gateway Timeout | The gateway has timed out. |
| 505 | HTTP Version Not Supported | The server does not support the "http protocol" version. |