HTTP Status Code

The Server issues an HTTP Status Code in response to a request of the client made to the server. Status code is a 3-digit integer. The first digit of status code is used to specify one of five standard classes of responses. The last two digits of status code do not have any categorization role.

The status codes are divided into 5 parts, as follows:

S. N.	Code and Description
1	**1xx: Informational Response** It is used to show that the request was received, and the process is continuing.
2	**2xx: Successful** It is used to show that the request was successfully received, understood, and accepted.
3	**3xx: Redirection** It is used to show that further action needs to be taken to complete the request.
4	**4xx: Client Error** It is used to show that the request contains bad syntax or cannot be fulfilled.
5	**5xx: Server Error** It is used to show that the server is failed to fulfill an apparently valid request.

HTTP status codes are extensible. The application of HTTP is not required o understand the meaning of all the registered status code. A list of all the status codes is given below:

1xx: Information

Message	Description
100 Continue	It is used to show that the client should continue with its request. The interim response informs the client that the request?s initial part has been received.
101 Switching Protocols	It is used to switches the server.
102 Processing	This code is used to show that the server has received and is processing the request. It indicates that no response is available yet.
103 Early Hints	This code is used to return the headers of some responses before the final HTTP message.

2xx: Successful

Message	Description
200 OK	This code is used to show that the request is OK.
201 Created	This code shows that the request has been fulfilled, which results in the creation of a new resource.
202 Accepted	This code shows that the request is accepted for processing, but not yet processed completely.

203 Non-authoritative Information	In the entity-header, the information is from a local third party copy. It is not from the original copy.
204 No Content	This code is used to show that the request is processed successfully by the server and not returning any content.
205 Reset Content	This code is used to tell the user agent to reset the document which sent this request.

3xx: Redirection

Message	Description
300 Multiple Choices	This code is used to indicate that the multiple options for the resource from which the client may choose.
301 Moved Permanentl y	This code shows that the URL of the requested resource has been changed permanently. In response, the new URL gives.
302 Found	This code is used to show that the requested page has moved temporarily to a new URL.
303 See Other	This code is used to show that the requested page can be found under another URL using the GET method.

304 No Modified	This code is used for caching purposes. It shows the client that the response has not been modified, so the client can continue to use the same response?s cached version.
305 Use Proxy	This code is used to show that using the proxy; the requested URL must be accessed, which is mentioned in the Location header.
306 Unused	In the previous version, this code is used. This response code is no longer used, and it is just reserved.
307 Temporary Redirect	This code is used to show that the requested page has moved temporarily to a new URL.

4xx: Client Error

Message	**Description**
400 Bad Request	This code is used to indicate that the server did not understand the request due to invalid syntax.
401 Unauthorized	In this code, the requested page needs a username and password.
402 Payment Required	This code reserved for future use.
403 Forbidden	This code is used to show that the access is forbidden to the requested page.

404 No Found	This code is used to show that the server cannot find the requested page.
405 Method Not Allowed	It shows that the request method is not supported for the requested resource.
406 Not Acceptable	It is used to show that the server can only generate a resource that the client does not accept.
407 Proxy Authentication Required	It is used to show that the client must first authenticate itself with the proxy.
408 Request Timeout	This code is used to show that the request took longer than the server was prepared to wait.

5xx: Server Error

Message	Description
500 Internal Server Error	This code is used to show that the server has encountered a situation, and it does not know how to handle it.
501 Not Implemented	This code shows that the request was not completed, and the server did not support the functionally required.
502 Bad Gateway	This code shows that the request was not completed, and the server received an invalid response from the upstream server.

503 Service Unavailable	This code shows that the request was not completed, and the server is temporarily overloading or down.
504 Gateway Timeout	It shows that the gateway has timed out.
505 HTTP Version Not Supported	This code is used to show that the server does not support the "http protocol" version.

REST API Architectural Constraints

REST stands for REpresentational State Transfer and API stands for Application Program Interface. REST is a software architectural style that defines the set of rules to be used for creating web services. Web services which follow the REST architectural style are known as RESTful web services. It allows requesting systems to access and manipulate web resources by using a uniform and predefined set of rules. Interaction in REST based systems happen through Internet’s Hypertext Transfer Protocol (HTTP).

A Restful system consists of a:

- client who requests for the resources.
- server who has the resources.

It is important to create REST API according to industry standards which results in ease of development and increase client adoption.

Architectural Constraints of RESTful API: There are six architectural constraints which makes any web service are listed below:

- Uniform Interface
- Stateless
- Cacheable
- Client-Server

- Layered System
- Code on Demand

The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

Uniform Interface: It is a key constraint that differentiate between a REST API and Non-REST API. It suggests that there should be an uniform way of interacting with a given server irrespective of device or type of application (website, mobile app).

There are four guidelines principle of Uniform Interface are:

- **Resource-Based:** Individual resources are identified in requests. For example: API/users.
- **Manipulation of Resources Through Representations:** Client has representation of resource and it contains enough information to modify or delete the resource on the server, provided it has permission to do so. Example: Usually user get a user id when user request for a list of users and then use that id to delete or modify that particular user.
- **Self-descriptive Messages:** Each message includes enough information to describe how to process the message so that server can easily analyses the request.
- **Hypermedia as the Engine of Application State (HATEOAS):** It need to include links for each response so that client can discover other resources easily.

Stateless: It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session. In REST, the client must include all information for the server to fulfill the request whether as a part of query params, headers or URI. Statelessness enables greater availability since the server does not have to maintain, update or communicate that session state. There is a drawback when the client need to send too much data to the server so it reduces the scope of network optimization and requires more bandwidth.

Cacheable: Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side. Client will return the data from its cache for any subsequent request and there would be no need to send the request again to the server. A well-managed caching partially or completely eliminates some client–server interactions, further improving availability and performance. But sometime there are chances that user may receive stale data.

Client-Server: REST application should have a client-server architecture. A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server, and server is someone who holds the resources and are not concerned with the user interface or user state. They can evolve independently. Client doesn’t need to know anything about business logic and server doesn’t need to know anything about frontend UI.

Layered system: An application architecture needs to be composed of multiple layers. Each layer doesn’t know any thing about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server. Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.

Code on demand: It is an optional feature. According to this, servers can also provide executable code to the client. The examples of code on demand may include the compiled components such as Java applets and client-side scripts such as JavaScript.