HTTP response status codes

HTTP response status codes indicate whether a specific **HTTP** request has been successfully completed. Responses are grouped into five classes:

1. **Informational responses** (100 – 199)
2. **Successful responses** (200 – 299)
3. **Redirection messages** (300 – 399)
4. **Client error responses** (400 – 499)
5. **Server error responses** (500 – 599)

Information responses(1xx)

This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. There are no required headers for this class of status code. Since HTTP/1.0 did not define any 1xx status codes, servers MUST NOT send a 1xx response to an HTTP/1.0 client except under experimental conditions.

A client MUST be prepared to accept one or more 1xx status responses prior to a regular response, even if the client does not expect a 100 (Continue) status message. Unexpected 1xx status responses MAY be ignored by a user agent.

Proxies MUST forward 1xx responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the 1xx response. (For example, if a proxy adds a "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

100 Continue
This interim response indicates that the client should continue the request or ignore the response if the request is already finished.

101 Switching Protocols
This code is sent in response to an Upgrade request header from the client and indicates the protocol the server is switching to.

102 Processing (WebDAV)
This code indicates that the server has received and is processing the request, but a response has yet to be available.

103 Early Hints
This status code is primarily intended to be used with the Link header, letting the user agent start preloading resources while the server prepares a response.

Successful responses

This class of status code indicates that the client's request was successfully received, understood, and accepted.

200 OK
The request succeeded. The result meaning of "success" depends on the HTTP method:

- **GET**: The resource has been fetched and transmitted in the message body.
- **HEAD**: The representation headers are included in the response without any message body.
- **PUT or POST**: The resource describing the result of the action is transmitted in the message body.
- **TRACE**: The message body contains the request message as received by the server.

201 Created
The request succeeded, and a new resource was created as a result. This is typically the response sent after POST requests or some PUT requests.

202 Accepted
The request has been received but has not yet been acted upon. It is noncommittal since there is no way in HTTP to send an asynchronous response later indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

203 Non-Authoritative Information
This response code means the returned metadata is not precisely the same as is available from the origin server but is collected from a local or a third-party copy. This is mostly used for mirrors or backups of another resource. Except for that specific case, the 200 OK response is preferred for this status.

204 No Content
There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new ones.

205 Reset Content
Tells the user agent to reset the document which sent this request.

206 Partial Content
This response code is used when the Range header is sent from the client to request only part of a resource.

207 Multi-Status (WebDAV)
Conveys information about multiple resources, for situations where multiple status codes might be appropriate.

208 Already Reported (WebDAV)
Used inside a <dav:propstat> response element to avoid repeatedly enumerating the internal members of multiple bindings to the same collection.

226 IM Used (HTTP Delta encoding)
The server has fulfilled a GET request for the resource, and the response is a representation of the result of one or more instance manipulations applied to the current instance.

Redirection messages

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required MAY be carried out

by the user agent without interaction with the user if and only if the method used in the second request is GET or HEAD. A client SHOULD detect infinite redirection loops since such loops generate network traffic for each redirection.

Note: previous versions of this specification recommended a maximum of five redirections. Content developers should be aware that there might be clients that implement such a fixed limitation.

300 Multiple Choices
The request has more than one possible response. The user agent or user should choose one of them. (There is no standardized way of choosing one of the responses, but HTML links to the possibilities are recommended so the user can pick.)

301 Moved Permanently
The URL of the requested resource has been changed permanently. The new URL is given in the response.

302 Found
This response code means that the URI of the requested resource has been changed *temporarily*. Further changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.

303 See Other
The server sent this response to direct the client to get the requested resource at another URI with a GET request.

304 Not Modified
This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.

305 Use Proxy
Deprecated

Defined in a previous version of the HTTP specification to indicate that a requested response must be accessed by a proxy. It has been deprecated due to security concerns regarding the in-band configuration of a proxy.

306 unused
This response code is no longer used; it is just reserved. It was used in a previous version of the HTTP/1.1 specification.

307 Temporary Redirect
The server sends this response to direct the client to get the requested resource at another URI with the same method that was used in the prior request. This has the same semantics as the 302 Found HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

308 Permanent Redirect
This means that the resource is now permanently located at another URI, specified by the Location: HTTP Response header. This has the same semantics as the 301 Moved Permanently HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

Client error responses

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents SHOULD display any included entity to the user.

If the client is sending data, a server implementation using TCP SHOULD be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

400 Bad Request
The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

401 Unauthorized
Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

402 Payment Required
Experimental
This response code is reserved for future use. The initial aim for creating this code was to use it for digital payment systems, however, this status code is used very rarely and no standard convention exists.

403 Forbidden
The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401 Unauthorized, the client's identity is known to the server.

404 Not Found
The server cannot find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 Forbidden to hide the existence of a resource from an unauthorized client. This response code is probably the most well-known due to its frequent occurrence on the web.

405 Method Not Allowed
The request method is known by the server but is not supported by the target resource. For example, an API may not allow calling DELETE to remove a resource.

406 Not Acceptable

This response is sent when the web server, after performing server-driven content negotiation, doesn't find any content that conforms to the criteria given by the user agent.

407 Proxy Authentication Required
This is similar to 401 Unauthorized but authentication is needed to be done by a proxy.

408 Request Timeout
This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers, like Chrome, Firefox 27+, or IE9, use HTTP pre-connection mechanisms to speed up surfing. Also, note that some servers merely shut down the connection without sending this message.

409 Conflict
This response is sent when a request conflicts with the current state of the server.

410 Gone
This response is sent when the requested content has been permanently deleted from the server, with no forwarding address. Clients are expected to remove their caches and links to the resource. The HTTP specification intends this status code to be used for "limited-time, promotional services". APIs should not feel compelled to indicate resources that have been deleted with this status code.

411 Length Required
Server rejected the request because the Content-Length header field is not defined and the server requires it.

412 Precondition Failed
The client has indicated preconditions in its headers that the server does not meet.

413 Payload Too Large

Request entity is larger than the limits defined by the server. The server might close the connection or return a Retry-After header field.

414 URI Too Long
The URI requested by the client is longer than the server is willing to interpret.

415 Unsupported Media Type
The media format of the requested data is not supported by the server, so the server is rejecting the request.

416 Range Not Satisfiable
The range specified by the Range header field in the request cannot be fulfilled. It's possible that the range is outside the size of the target URI's data.

417 Expectation Failed
This response code means the expectation indicated by the Expect request header field cannot be met by the server.

418 I'm a teapot
The server refuses the attempt to brew coffee with a teapot.

421 Misdirected Request
The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.

422 Unprocessable Entity (WebDAV)
The request was well-formed but was unable to be followed due to semantic errors.

423 Locked (WebDAV)
The resource that is being accessed is locked.

424 Failed Dependency (WebDAV)

The request failed due to the failure of a previous request.

425 Too Early
Experimental
Indicates that the server is unwilling to risk processing a request that might be replayed.

426 Upgrade Required
The server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol. The server sends an Upgrade header in a 426 response to indicate the required protocol(s).

428 Precondition Required
The origin server requires the request to be conditional. This response is intended to prevent the 'lost update' problem, where a client GETs a resource's state, modifies it, and PUTs it back to the server, when meanwhile a third party has modified the state on the server, leading to a conflict.

429 Too Many Requests
The user has sent too many requests in a given amount of time ("rate limiting").

431 Request Header Fields Too Large
The server is unwilling to process the request because its header fields are too large. The request may be resubmitted after reducing the size of the request header fields.

451 Unavailable For Legal Reasons
The user agent requested a resource that cannot legally be provided, such as a web page censored by a government.

Server error responses

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include an entity

containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

500 Internal Server Error
The server has encountered a situation it does not know how to handle.

501 Not Implemented
The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

502 Bad Gateway
This error response means that the server while working as a gateway to get a response needed to handle the request, got an invalid response.

503 Service Unavailable
The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-friendly page explaining the problem should be sent. This response should be used for temporary conditions and the Retry-After HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care of the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.

504 Gateway Timeout
This error response is given when the server is acting as a gateway and cannot get a response in time.

505 HTTP Version Not Supported
The HTTP version used in the request is not supported by the server.

506 Variant Also Negotiates

The server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper end point in the negotiation process.

507 Insufficient Storage (WebDAV)
The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request.

508 Loop Detected (WebDAV)
The server detected an infinite loop while processing the request.

510 Not Extended
Further extensions to the request are required for the server to fulfill it.

511 Network Authentication Required
Indicates that the client needs to authenticate to gain network access.

Why Restful API

One of the most popular types of API is REST or, as they're sometimes known, RESTful APIs. There are many benefits of REST or RESTful APIs — they are designed to take advantage of existing protocols. While REST - or Representational State Transfer - can be used over nearly any protocol, when used for web APIs it typically takes advantage of HTTP. This means that developers have no need to install additional software or libraries when creating a REST API. One of the key advantages of REST APIs is that they provide a great deal of flexibility. Data is not tied to resources or methods, so REST can handle multiple types of calls, return different data formats and even change structurally with the correct implementation of hypermedia. This flexibility allows developers to build an API that meets your needs while also meeting the needs of very diverse customers.

There are 6 key constraints to think about when considering whether a RESTful API is a right type of API for your needs:

- Client-Server: This constraint operates on the concept that the client and the server should be separate from each other and allowed to evolve individually.
- Stateless: REST APIs are stateless, meaning that calls can be made independently of one another, and each call contains all of the data necessary to complete itself successfully.
- Cache: Because a stateless API can increase request overhead by handling large loads of incoming and outbound calls, a REST API should be designed to encourage the storage of cacheable data.
- Uniform Interface: The key to the decoupling client from the server is having a uniform interface that allows independent evolution of the application without having the application's services, or models and actions, tightly coupled to the API layer itself.
- Layered System: REST APIs have different layers of their architecture working together to build a hierarchy that helps create a more scalable and modular application.
- Code on Demand: Code on Demand allows for code or applets to be transmitted via the API for use within the application.

Unlike SOAP, REST is not constrained to XML but instead can return XML, JSON, YAML, or any other format depending on what the client requests. And unlike RPC, users aren't required to know procedure names or specific parameters in a specific order.

One of the disadvantages of RESTful APIs is that you can lose the ability to maintain a state in REST, such as within sessions. It can also be more difficult for newer developers to use.

It's important to understand what makes a REST API RESTful, and why these constraints exist before building your API.