PUT

HTTP PUT is like HTTP POST except the URI does not handle the request, it is the request. It creates a new resource or replaces a representation of the target resource with the request payload.

> The URI in a POST request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request – the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to some other resource.
>
> Fielding, et al. ,RFC2616: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.6

The difference between `PUT` and `POST` is that `PUT` is idempotent; calling it once or several times successively has the same effect i.e., that is no side effect, where as successive identical `POST` requests may have additional effects, akin to placing an order several times.

Moreover PUT:

- URI identifies an entity (file, db entry . . .).
- Arguments are stored in the URI query string or HTTP headers.
- request body contains the entire entity, it essentially overwrites or creates a new entity with that content.

PUT can be used for:

- Create a new entity at the URI
- Replace an existing entity at the URI
- Add/replace an entry to a DB
- Entity can be retrieved later with GET and the same URI

DELETE

Like HTTP POST except the URI does not handle the request, it is the request, a request to delete the entity at that URI.

> 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.
>
> Fielding, et al. ,RFC2616: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.7

Moreover DELETE:

- URI identities an entity (file, db entry. . .).
- Arguments are stored in the URI query string or HTTP headers
- Request body is usually empty
- Response body is usually empty
- Basically delete this entity, if I GET it again it should response with a 404 error if I didn't PUT or POST at that same URI.

DELETE can be used for:

- Delete the entity at the URI
- Delete a file on the server's file system
- Remove an entry in a DB
- Entity cannot be retrieved later with GET and the same URI

WebDAV

WebDAV is an extension of HTTP that allows clients to perform remote Web content authoring operations. Basically FTP but for the web

- Let's you create and upload to a URI using HTTP PUT
- Download from a URI using HTTP GET
- Delete an entity at a URI and the URI using HTTP DELETE
- Make directories and folders using a new HTTP command MKCOL for MaKeCOLlection

User Agent

A user agent is the client or browser that is accessing the web services or server. It is a HTTP header that is part of the HTTP request that is sent to the server. `User-Agent` is the header's name.

> **Note**
> Not all HTTPs are browser, cURL is not a web browser but it is a web client. Therefore it will provide a user agent to the server.

The user agent will tell a lot about the client, for web browsers they might show

- Browser i.e., Chrome, Firefox, etc
- Version
- Operating system
- Machine architecture
- Rendering engine
- etc

Mobile version will also have their own separate user-agent, even the Nintendo Switch will have their own user agent.

HTTP Status Codes

There are five different sets of status codes that can be sent as a response.

Code Range	Status Type
`1XX`	Information codes
`2XX`	Success codes
`3XX`	Redirection codes
`4XX`	Client Error codes
`5XX`	Server Error code

1XX: Information codes

These codes are use when they are negotiating a transaction. There are many but in this we are going to look at 2.

100 Continue

Used in multipart and uploads, specifically tells the client to send the request body/data. The server has the choice to accept the request or not and with the 100 code it has decided to accept it

101 Switching Protocols

This is a rare status code where it provides `Upgrade` as a header. This specify the new protocol to change to after the blank line at the end of the header. An example situation could be switching from HTTP/1.1 unencrypted to HTTP/2.0 but doesn't really happen as HTTP/2.0 only works on encrypted connections for browsers.

2XX: Success codes

These are success codes, meaning that the transactions between server and client was a success.

200 OK

The request was a success depending on the client's request.

- GET: sends an entity for the requested URI
- HEAD: sends headers for the entity for the requested URI
- POST: sends an entity describing the result of the POST
- TRACE: sends back the entity it received

201 Created

Request succeeded and a new entity was created and exists (e.g. PUT)

202 Accepted

Like `200 OK` but the server's not done with it yet. For example, you asked the server to perform a calculation and its working on it.

203 Non-Authoritative Information

This is a rare code used with proxies.

204 No Content

Request succeeded but the server is only sending headers and no entity or response body.

205 Reset Content

Rare status code, like `204 No Content` but the browser should clear the form/page.

206 Partial Content

Used to resume downloads. The client request with a `Range` header to continue a big download that was interrupted. The server responds with a `206 Partial Content` code and a header named `Content-Range` is returned indicating what part it's sending.

3XX: Redirection Codes

Redirects is a very powerful for HTTP. It is a cheap form of abstraction, provides work-around for browser/protocol issues, redirects HTTP to HTTPS, redirect request from old URIs to new URIs. It provides load balancing, separate dynamic and static content onto two different servers, allows more than one server behind the scenes. It allows URLs to be shorted and allows websites to be reorganized but still allow old URIs to keep working (fighting link-rot).

300 Multiple Choices

This is a rare response that provides a list of choices and the user or the browser chooses one. An example could be selecting a language for a page.

301 Moved Permanently

Go to the URI mentioned in the `Location` header, and don't ask again. The new location should be cached and the URI in the location bar automatically changes.

302 Found

This is a temporary redirect and the client should GET the URI mentioned int he `Location` header and display that response instead. This is useful for load-balancing.

Note

The URI in the location bar stays the same, the user doesn't know that it is in a different location.

303 See Other

Exists to solve the form-POSTing problem where POSTing the same post will cause duplicate post in the forum. GET the URI in the `Location` header and don't save the redirect in your cache, you can keep making POSTs to the URI that gave you a 303 code. The URI in the location bar changes.

304 Not Modified

The browser can make a conditional GET request for URIs that it has cached, asking the server to send the entity only if it's changed since the time it was cached or a specific version. There are two headers for this response:

- `If-None-Match` - Followed by a list of etags (like git tags) will get a 200 OK only if it has a new version not in the list
- `If-Modified-Since` - Followed by a date and time, will only get a 200 OK only if it has a newer version since that time.

The response will have no body or entity.

305 Use Proxy

Rare response, try the proxy server specified by the `Location` header.

307 Temporary Redirect

Go to the URI mentioned in the `Location` header. Because it is temporary the client can keep making requests to the URI you originally requested in case the sever needs to redirect you somewhere else next time. Cache the redirection using standard caching headers and rules. The URI in the location bar is updated.

4XX: Client Error codes

It's your fault!!!!

Basically anything that is caused by even if you didn't know that that server couldn't do or support that.

400 Bad Request

The server can't read your garbage. Don't send it again. Basically your request is malformed.

401 Unauthorized

You have to send authentication information to see this URI. Header and entity(body) explains to the browser and user how to log in. Mostly useful for HTTP `Authorization:` header authentication.

402 Payment Required

Pay before moving on. Supposedly reserved but some services use it anyway like MobileMe, Google APIs use it. Youtube will use it to force you to solve a CAPTCHA.

403 Forbidden

The web server will never respond to this request, no matter who you log in as. Maybe it could answer your request buy an administrator disabled that ability.

404 Not Found

Self Explanatory. You've got the wrong resource or path, can't find the thing.

405 Method not allowed

Whatever HTTP method you used doesn't work on this URI.

406 Not Acceptable

The server cannot respond in way that matches your request's `Accept` header line. You ask for a type that the server cannot handle, and it's your fault for not knowing.

407 Proxy Authenticating Required

We're not going to proxy your request till you authenticate.

408 Request time out

You took too long to send your request, we're not going to service you. Try again but faster next time. This is to prevent a Slowloris attack where the attacker will take it sweet ass time trying to take resources away from other users.

409 Conflict

The request is in conflict. Often used with `PUT` requests. Basically there is a concurrency issue at the URI.

410 Gone

Yeah it was here, but it ain't coming back. Don't even try again. Meaning this request should be cached.

411 Length Required

I can't service a request like `POST` without a `Content-Length:` header.

412 Precondition Failed

Header information wouldn't be what you wanted it to be so I won't process the request. An example could be modified too recently, so don't allow PUT to succeed to modify it again.

413 Request Entity Too Large

Sending an entity (POST, PUT, . . .) that's bigger than the server can handle. The client should know better.

414 Request-URI Too Long

The web server cannot process the URI because it's too long and will overfill the buffer. Apache web servers is limited to 4000 to 8192 bytes depending on the version. nginx web server places a limit on total length of HTTP headers + requested URI.

415 Unsupported Media Type

Uploading (POST, PUT, . . .) using a format the server doesn't understand. Like posting pictures formatted in JPEG2000 or some other esoteric format.

416 Request Range Not Satisfiable

You sent a `Range:` header to get just part of a file but the part you asked for doesn't make sense. An example could be asking to resume a download that was interrupted for a 1 Mib file at 1.3 Mib

417 Expectation Failed

The server cannot meet the `Expect:` header. An example could be that the client send a `Expect: 100 Continue` while POSTing multipart/form-data, but the server can't do that.

418 I'm a teapot

Indicates that the server refuses to brew coffee because it is a teapot. Was an April Fools' joke that was built into the standard.

422 Unprocessable Entity

Indicates that the server understood the Content-Type and the syntax of the entity (request body) is correct but that it was unable to process it.

426 Upgrade Required

Indicates that the server requires use of HTTP 2 or later.

428 Precondition Required

Indicates that the client needs to send a request with an `If-` header.

429 Too Many Requests

Indicates that the client has sent too many request in a short period of time

431 Request Header Fields Too Large

Indicates that the client has sent request headers that are too long.

451 Unavailable For Legal Reasons

Indicates that the server could service the request if it wasn't illegal.

Note

The 451 code was used because it's a reference to the book Fahrenheit 451. People also call this code 451 censorship.

5XX: Server Error codes

It's the server's fault

Basically anything that deals with the server crashing, or a script on the server crashing, etc.

500 Internal Server Error

Server side software encountered some kind of error

501 Not Implemented

The server can't fulfill that request (such as a n HTTP PUT) because it doesn't even know what HTTP PUT is.

502 Bad Gateway

The server talks to anther HTTP server to fulfill this request and that other server isn't working.

503 Service Unavailable

The service is temporarily down. Something's broken and we'll bring it back up eventually.

504 Gateway Timeout

The server talks to another process to fulfill this request and that other process isn't responding fast enough. Very common when a webapp is overloaded. Similar to 502, except in this case the packets between the reverse proxy and the origin webserver are just vanishing.

505 HTTP Version Not Supported

Your request used the wrong HTTP version. A version the server no longer supports. For example Twitter doesn't let you do HTTP/1.0 requests anymore.

511 Network Authentication Required

Used by *captive portals* to tell the web browser that it should show the user a login page for the network. Your request never made it to the web server until after you've authenticated on the network.

Status code summary

So what codes should you use? Some suggest that business logic errors (software error reporting) should be done with the client without codes. Some suggest that you should use HTTP status code. In general the decision should be decided if the application is user facing and how you should handle it for your audience.

- Do you need to talk to the User Agent or to the user?
- Example: many websites let you log in with cookie-based authentication. There's no standards-compliant way to use a response code, e.g. 401 Unauthorized to tell the user to log in this way. So it may be best to use a 302 or 307 redirect to send the user to the login page.

HTTP Headers

There are a lot of headers within the HTTP specification. They all provide additional context and information to the request and response messages to and from the server. These headers have an effect on:

- Authentication
- Caching
- Encoding
- Partial downloading
- Content type
- More...

You can look them up on RFC 2616 Section 14.

HTTP Request Headers

These headers are only allowed in a request message to the server.

Accept

Specifies the kind of media the client can handle

Syntax	Meaning
`<MIME_type>/<MIME_subtyp`	A single precise MINE Type like `text/html`
`<MIME_type>/*`	A MIME type, but without a subtype
`*/*`	Any MIME type
`;q=`	Weight to add on a MIME type

Accept-Charset

Specifies character encodings the client can handle.

Accept-Encoding

Specifies compression formats the client can handle. Examples

- gzip
- compress
- deflate

Access-Control-Request-Headers: Content-Type

Lets the browser ask the server if JS is allowed to make requests with those headers

Access-Control-Request-Method: POST

Lets the browser ask the server if JS is allowed to make e.g. POST requests

Authorization

Syntax. `Authorization: <auth-scheme> <authorisation-parameters>` Where: `<auth-scheme>` is the authorization scheme that defines how the credentials are encoded.

The user agent is sending a username and password or other kind of credentials to the server (rare, usually cookies are used instead)

Cache-Control

Asks the server/proxy not to send data thats been sitting in its cache too long. The directives could be in seconds like `max-age=60`.

Content-Disposition

Syntax. `Content-Disposition: attachment; filename="filename.jpg"`.

Tells the server what the name/filename of the form data being uploaded when POSTing multipart/form-data.

Cookie

The user agent is sending cookies (stored key-value pairs) relevant to the server. The cookies were previously sent to the user agent to store by the server or JS.

DNT

Do Not Track, largely ignored because webservers wants to track you regardless. The user prefers not to be tracked over receiving personalized content.

Expect: 100-continue

The user agent is expects the server to respond with `100 Continue`.

Forwarded

Syntax. `Forwarded: by=<identifier>; for=<identifier>; host=<host>; proto=<http|https>`. Where:

<identifier>	An identifier disclosing the information that is altered or lost when using a proxy.
by=<identifier>	The interface where the request came in to the proxy server.
for=<identifier>	The client that initiated the request and subsequent proxies in a chain of proxies.
host=<host>	The Host request header field as received by the proxy.
proto=<http|https>	Indicates which protocol was used to make the request (typically "http" or "https").

Used by (reverse) proxies to tell the server who made the original request, over what protocol, and what the original Host header was.

From

Syntax. `From: <email>`.

Email address of the person making the requests. Example: bot owner, so people can contact them about their bot if it misbehaves.

Host

Syntax. `Host: <host>:<port>`.

The hostname (and sometimes port) of the website the user agent is trying to connect to. When a single server or proxy is handling requests for many different websites, it needs to know which site the request was made to. Otherwise it only can differentiate by IP address, but server/proxy usually has only one public IP address. Required in HTTP/1.1 and later for all requests. If Host: is missing the server may respond with 400 Bad Request.

If-Match

Asks the server to send the content only if the ETag(s) matches the specified string.

If-None-Match

Asks the server to send the content only if the ETag(s) **doesn't** match the specified string.

If-Modified-Since

Asks the server to send the content only if it's changed recently.

If-Unmodified-Since

Asks the server to accept the request only if it hasn't changed recently.

If-Range

Only resume download if I'm still going to download the same version, otherwise start over.

Origin

Tells the server where the JS code making this request came from.

Proxy-Authorization

Log into forward proxy server.

Range

Resume download at the specified point.

Referer

Tells the server what URI you were viewing that caused you to make the current request Shows what page had the link you clicked to get to the current page Destroys privacy Lets server admins know, for example, that people are finding their page on Google, or Twitter, etc.

TE

Like Accept-Encoding but used with a proxy.

Upgrade-Insecure-Requests

Asks the server to send a redirect to the HTTPS version of the page.

User-Agent

Tells the server what version of browser/client is making the request.

HTTP Server Headers

Access-Control-Allow-Credentials: true

Whether to allow JS code running in the browser to make requests with cookies.

Access-Control-Allow-Headers

Whether to allow JS code running in the browser to make requests with extra headers specified in the header.

`Access-Control-Allow-Headers: [<header-name>[, <header-name>]*]`

Access-Control-Allow-Methods

What methods the server will allow JS code running in the browser to make.

`Access-Control-Allow-Methods: <method>, <method>`

Access-Control-Allow-origin

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: <origin>
Access-Control-Allow-Origin: null
```

Only allow JS that came from a certain host to make requests to this server

Access-Control-Expose-Headers

`Access-Control-Expose-Headers: [<header-name>[, <header-name>]*]`

Whether to allow JS code running in the browser to see headers

Access-Control-Max-Age

`Access-Control-Max-Age: <delta-seconds>`

How long the browser can remember the other access-control-headers

Age

`Age: <delta-seconds>`

(reverse) proxy cache that's e.g. 24 seconds old

Allow

`Allow: <http-methods>`

Tells the user agent what HTTP methods the server supports

Connection: close

Tells the server to close the connection after it's done sending the content

Content-Disposition

Tells the browser to prompt the user to save the content with a default filename, instead of displaying the content.

Similar to the Request version of the header.

Content-Encoding

Tells the browser to decompress the content before using it and what format its compressed in.

Similar to the Request version of the header.

Content-Language

Tells the browser what natural language the content its sending is in

Content-Length

Tells the client how many bytes of content to expects.

Content-Location

Tells the client where it can find the content that matches the Accept, Accept-* headers it sent. Tells the client where it can find the content that it created using, e.g. POST.

Content-Range

Tells teh client what bytes of the requested content the server is sending, out of the total length in bytes.

`Content-Range: <unit> <range-start>-<range-end>/<size> Content-Range: <unit> <range-start>-<range-end>/* Content-Range: <unit> */<size>`

Content-Security-Policy

Used to combat Cross Site Scripting attacks. Example: Restrict the places that the User Agent should fetch and run JS from.

Content-Security-Policy-Report-Only

Used to debug `Content-Security-Policy`

Content-Type

Tells the user agent what kind of media the server's sending

ETag

Version number/name of the content. If it changes it means the content has changed. Example: a SHA1 hash of the content.

Expect-CT

Tells the web browser to double-check the TLS certificate with public certificate log service

`Expect-CT: report-uri="<uri>", enforce, max-age=<age>`

Date

The date and time the server sent the content.

Expires

Tells the user agent or caching proxy to cache a response only until the specified date and time.

Last-Modified

Tells the user agent or caching proxy when the content last actually changed.

Keep-Alive

`Keep-Alive: [timeout=<sec>] [max=<int>]`

Tells the client not to send more than `<int>` requests on this connection or to let it idle for more than `<sec>` seconds.

Location

Tells the client where to redirect to for 300 series status codes.

Proxy-Authenticate, Proxy-Authorization

Ask client to log into proxy server.

Public-Key-Pins

Tells the client not to only accept one of the two specified certificates from the webserver for the next certain amount of time. Dangerous: can lock people out of being able to view your site for a long time Disabled in Chrome. This is also a depreciated header.

Referrer-Policy: no-referrer-when-downgrade

Webserver tells the client not to send as much referrer information to the next webserver.

Retry-After

Webserver tells the client to wait a certain amount of time before retrying after a 503 Service Unavailable, 429 Too Many Requests or how long to delay before following a 3xx redirect

Server

Tells the client the server software and version, like User Agent but for servers.

Set-Cookie

Webserver tells the client to store a cookie (key-value pair) in its cookie jar. The cookie will be stored and sent back to the webserver every time the client makes a request, using the Cookie: header. But only if the security restrictions are met: only over HTTPS, not with requests made by JS, not with requests initiated by code that came from a different website, delete the cookie after a day...

Transfer-Encoding

Like Content-Encoding but used by a proxy. If `chunked` is specified data will be sent in chunks. Replaced by HTTP/2 for that encoding only.

Strict-Transport-Security

Don't contact this server or any subdomains without TLS for a specified period of time.

Trailer: Expires

Tells the client there will be additional headers after the content is sent. HTTP/1.1 requires Transfer-Encoding: Chunked for this to work. HTTP/2 doesn't work in all browsers.

Vary: Accept-Language

Tells a cache that this page is one of multiple versions of the page based on the HTTP request headers listed.

Via

Records the proxy chain a request or response went through.

`Via: [ <protocol-name> "/" ] <protocol-version> <host> [ ":" <port> ] Via: [ <protocol-name> "/" ] <protocol-version> <pseudonym>`

WWW-Authenticate
Sent with 401 Unauthorized to tell the user agent/user how to authenticate Browser will pop up a dialog box with "What is the password" on it

Warning
There are a whole bunch of warning codes

Code	Reason	Explanation
110	Response is Stale	Warns the client that the content was in the cache for too long.
111	Revalidation Failed	Warns the client that the content was in the cache but the original server is down.
112	Disconnected Operation	Warns the client that the content was in the cache but the network is down.
113	Heuristic Expiration	Warns the client that the content was in the cache over 24 hours.
199	Miscellaneous Warning	Warns the client about something.
214	Transformation Applied	Warns the client that the content was changed in some way by the proxy.
299	Persistent Warning	Warns the client about something the proxy doesn't expect to change anytime soon

X-Content-Type-Options: nosniff
Tells the client not to try to figure out the Content Type (MIME Type) of the content on its own, e.g. based on contents.

X-DNS-Prefetch-Control: off
Tells the browser not to resolve hostnames in links before the user clicks on them.

X-Frame-Options: deny
Tells the browser not to show the page in a <frame>, <iframe>, <embed> or <object>.

Custom HTTP Headers
Used to have use X- headers, but X- headers are now deprecated. You can add whatever headers you want as long as they don't conflict with standard headers.

Authentication
Multiple strategies for authentication using HTTP exist:

- Cookies
  - Session-based (stateful)
  - Token-based (stateless)
  - Easy to mess up and create a security problem
- Authorization:
  - Basic (old, insecure)
  - JWTs (JSON Web Token), RFC 7519 <- this is the most common

Session Cookies
1. User browses to site
   - Doesn't have cookie yet so browser doesn't send any Cookie: headers
2. Server responds with cookie
   - Generates a large random number, ex: 86e7eca9-d484-48ad-98ed-e060541b7870
   - Records session number in server's database
   - Responds with Set-Cookie header, ex: Set-Cookie: s=86e7eca9-d484-48ad-98ed-e060541b7870; Secure; HttpOnly; SameSite=strict
   - Responds with redirect or page allowing the user to log in

HTTP proxies
Forward Proxies
- Rare
- Forward HTTP requests for you
- You set it up
- Example: Watch US Netflix from Canada

Reverse Proxies
- Common
- Forward HTTP requests for the server
- Set up by the people managing the server
- Example: Cloudflare CDN

Reverse proxies are used for

- Caching
- CDN: Keep content nearer the user
- Load balancing
- TLS
- Fail-over

Generally there are multiple layers of proxy servers that communicate between the user agent and the application server. The application server's database could also be on a separate system adding another layer of latency.

Review
URI. Universal Resource Identifier, the location of a resource on a "webserver"

HTTP Methods. Here are the four basic ones you should know:

- GET - Get (potentially) dynamic information from a URI.
- POST - Run search, apply forms, append, and change data URI.

- PUT - Store the entity at that URI, full replacement (overwrite) or insertion at the URI.
- DELETE - Delete the resource at that URI

URL.

- Has a syntax and schema.
- Can be absolute or relative for both service and path.
- Can query using (`?key1=value1&key2=value2&...`) in the URL but is limited, long queries should be done in a POST.
- Queries could just be a string as well
- Fragments are accessed using `#` like accessing a section on a page

Request and Response.

- HTTP 1.1 requests requires the Request Line i.e., `METHOD path HTTP/1.1` and the `Host:` header.
- HTTP 1.1 responses requires the Response Line i.e., `HTTP/1.1 status_code`.
- All headers are ended with an empty `CRLF` or
  `r`
  `n`.
- HTTP header names follow capitalized snake case with the space is `-` and the first letter in each word is capitalized.

Status Codes.

- 1xx are informational
- 2xx are success codes
- 3xx are redirection codes
- 4xx are client side errors
- 5xx are server side errors
- There are whole bunch just look them up

Authentication.

- Cookies to store auth information, easy to mess up and creates a security problem. They can be session-based with a state or token-based that is stateless.
- Authorization tokens are more secure and commonly used.

Proxies.

- Forward proxies are client side that you have to setup and forward HTTP requests for you
- Reverse proxies are server side that does a lot of things, load balancing, caching, CDN which keeps the content closer to the user, etc.