HTTP

André Restivo

Index

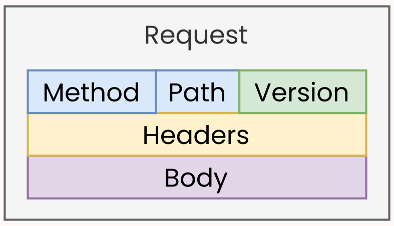
Introduction

HTTP

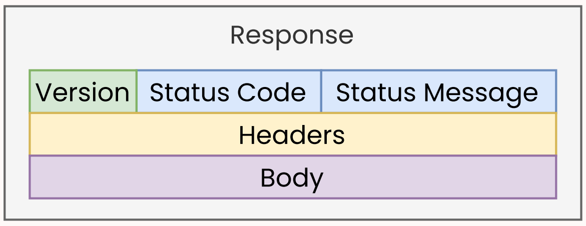
- **H**yper **T**ext **T**ransfer **P**rotocol.
- **Application-layer** protocol for transmitting hypermedia documents.
- **Client-server** model.
- **Stateless** protocol.

How does it work?

1. The browser wants a **resource**. Either the user typed an URL, an HTML needs other resources, a link was followed, a form was submitted...
2. The browser establishes a TCP (most of the time) **connection** to the server.
3. The browser sends a **request**:



4. The server returns a **response**:



History

- **HTTP/0.9** (1991) Only **GET** method. No headers.
- **HTTP/1.0** (1992–96) Files of different types. **HEAD** and **POST**.
- **HTTP/1.1** (1995–97) Reuse connections. **Host** header.

Since then, the **HTTP 1.1** protocol evolved by adding new headers.

- **HTTP/2.0** (2014–15) A major revision of the **HTTP** network protocol.
- **HTTP/3** (2019–) A HTTP mapping over QUIC (a general-purpose transport layer network protocol designed by Jim Roskind at Google).

Resources

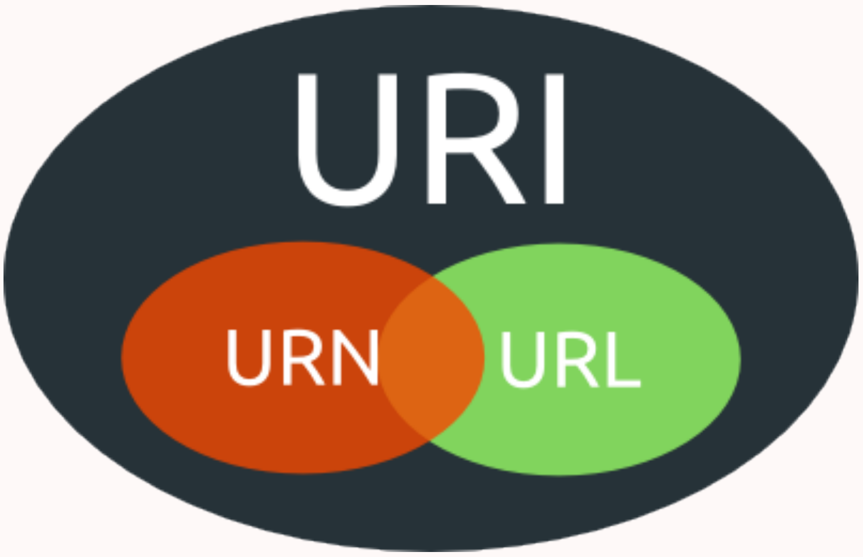
- References:
  - RFC2616: The official specification.
  - Mozilla Developer Network (MDN) Reference.
- Tools:
  - Playing with cURL.

URIs and URLs

URI

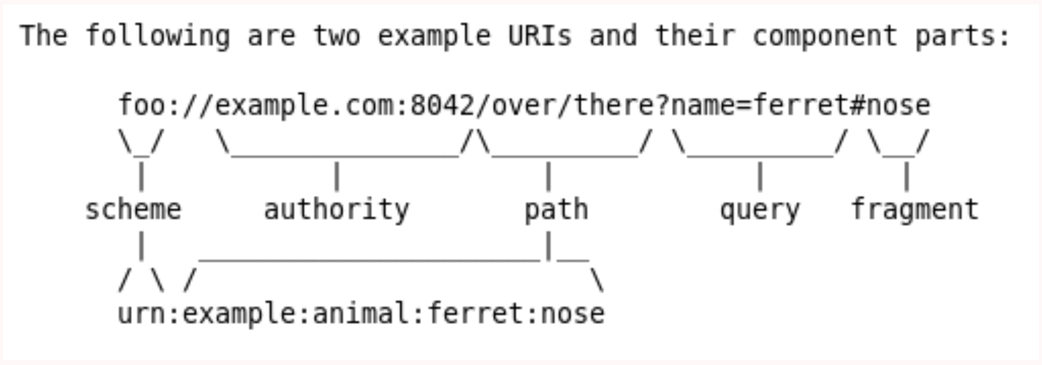
Uniform **R**esource **I**dentifier

- An identifier is an object that can act as a **reference** to something that has an **identity**.
- In the case of a URI, the object is a sequence of characters with a restricted **syntax** RFC3986.
- A URI can be further classified as a locator (**URL**), a name (**URN**), or both.
- URI components: scheme, authority, path, query, fragment



URN

Uniform **R**esource **N**ames are intended to serve as **persistent**, **location-independent**, resource **identifiers** RFC2141.



```
The following are two example URIs and their component parts:

      foo://example.com:8042/over/there?name=ferret#nose
      \_/   \______________/\_________/ \_________/ \__/
       |           |            |            |        |
    scheme     authority       path        query   fragment
       |   _____________________|__
      / \ /                        \
      urn:example:animal:ferret:nose
```

Source: RFC3986

URL

Uniform **R**esource **L**ocator

URL refers to the **subset** of URI that identifies resources via a representation of their primary access mechanism (e.g., their network *location*), rather than identifying the resource by name or by some other attribute(s) of that resource.

A Uniform Resource Name (URN) functions like a person’s **name**, while a Uniform Resource Locator (URL) resembles that person’s unique **address**.

HTTP URL

HTTP URL

Every **HTTP URL** consists of the following, in the given order:

- the **scheme** name (or protocol, *i.e.*, HTTP or HTTPS)
- a colon (:), two slashes (//)
- a **host** (domain name or IP address)
- optionally a colon (:) followed by a **port** number
- the full **path** of the resource
- optionally a **query** string
- optionally a **fragment** identifier

scheme://domain:port/path?query_string#fragment_id

Scheme Name

- For HTTP connections the scheme name can be either **HTTP** or **HTTPS**.
- **H**yper**t**ext **T**ransfer **P**rotocol **S**ecure (HTTPS) is just HTTP on top of the **SSL/TLS** protocol.

ℹ http://

Hostname

Either a domain name or an IP address.

> ℹ www.google.com

> ℹ 127.0.0.1

Port

- The **default port** for a HTTP server on a computer is port **80**.
- Others are also normally used: 8080, 8000.
- The port number can be **omitted** from the URL if it is the default one.

:80

Path

- The full path of the resource.
- A sequence of segments that are separated by slashes.
- **May** resemble or map exactly to a file system path **but not necessarily**.

> /somewhere/on/this/server.php

Query String

- The query string contains **data** to be passed to software running on the server.
- It may contain **name/value pairs** separated by ampersands.

?first_name=John&last_name=Doe

Fragment Identifier

- The fragment identifier, if present, specifies a **part** or a **position** within the overall resource or document.
- If used with HTML, represents an element in the page identified by its **id**.

ℹ #content

HTTP Request

Request

The first line contains a request **method** followed by its parameters:

- the **path** of the document (an absolute URL without the protocol and the domain name).
- the HTTP protocol **version** used.

```
GET /~arestivo/index.php HTTP/1.1
```

The subsequent lines each represent a specific HTTP **header**.

The final block is the optional **data block**. It's separated from the headers by a **blank line** and contains further data. Mainly used by the PUT, POST and PATCH methods.

Examples

A **GET** request:

```
GET /search.php?name=john HTTP/1.1
Host: www.example.com
Accept-Language: pt
```

A **POST** request:

```
POST /path/save.php HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded

name=John%20Doe&username=johndoe
```

HTTP 1.1 **requires** the Host header.

Methods

- The request method indicates the **action** to be performed by the server.
- They all have a **semantic** meaning but it is up to the developer to enforce that meaning.
- The HTTP/1.1 standard defines **nine** methods:
  - **GET**, **HEAD**.
  - **POST**, **PUT**, **DELETE**, **PATCH**.
  - **OPTIONS**, **TRACE**, **CONNECT**.
- Other standards can add extra methods.
- HTML links always use the **GET** method, while HTML forms can use the **GET** or **POST** methods.

Safe Methods

A **safe** method is a method that **doesn't have** any **side effects** on the server:

- **GET**: Requests a **representation** of a resource identified by the request URI. The **request** should **not include** any **data**.
- **HEAD**: Identical to GET but **only** requests the **headers** of the response. The **server** should **not** send any **data**. Used to get data about a resource without actually getting the resource.

All HTTP servers **must implement** these two methods. All other methods are optional.

Idempotent Methods

An **idempotent** method is a method where the **side effects** on the server of **several identical** requests are the **same** as the **side effects** of a **single request**.

- **PUT** Requests that the representation of a resource be **stored** at the supplied URI. Can be used to **create** or **replace** a representation.
- **DELETE** Request that the resource identified by the URI be **deleted**.
- **HEAD** and **GET** are also idempotent.

POST

- The **POST** method requests that the representation of a resource be **stored** at the supplied URI. Can be used to **create** or **replace** a representation.
- The difference between **POST** and **PUT** is that **POST** is **not idempotent**. Each identical call can have additional effects, *e.g.*, placing the same order multiple times.

POST/PUT Body

- The **type of body** is controlled by the Content-Type header (more on headers later.)

- In **HTML forms**, we can change this header using the enctype attribute on the form elements, or the formenctype of the button element.

Possible values for the **Content-Type** header in HTML forms:

- `application/x-www-form-urlencoded`: url-encoded **key-value tuples** separated by *& and with a* =* between key and value.
- `multipart/form-data`: Each value is sent in a **separate body** part with a Content-Disposition header. The way to send binary data.
- `text/plain`: A **single text** value.

POST/PUT Body Examples

```
POST /register.php HTTP/1.1
Host: foo.example
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

username=johndoe&field2=strongpassword123
```

```
POST /upload.php HTTP/1.1
Host: foo.example
Content-Type: multipart/form-data;boundary="bound

--boundary
Content-Disposition: form-data; name="description

An image of a dog.
--boundary
Content-Disposition: form-data; name="image"; fil

(binary bytes of the image)
--boundary--
```

OPTIONS

- The **OPTIONS** method requests communication options for a given URL or the entire server.
- Used in **preflight requests** in CORS (more on this later).

Example **OPTIONS** response:

```
HTTP/1.1 204 No Content
Allow: OPTIONS, GET, HEAD, POST
Cache-Control: max-age=604800
Date: Thu, 13 Oct 2016 11:45:00 GMT
Server: EOS (lax004/2813)
```

Other Methods

Other not so common/important methods:

- **TRACE**: Performs a loop-back test.
- **CONNECT**: Can be used to open a tunnel.
- **PATCH**: Applies partial modifications to a resource.

All methods as defined in *RFC 2616*.

HTTP Response

Status Line

The **status line** is the **first line** in the **response** message. It consists of **three** items:

- The HTTP **version number**.
- A **status code** — a three-digit code indicating if the request has been successful.
- A **reason phrase** (a non-authoritative, human-readable text that summarizes the meaning of the status code).

```
HTTP/1.0 200 OK
```

Responses can be grouped into five categories: **informational** (1xx), **success** (2xx), **redirection** (3xx), **client error** (4xx) and **server error** (5xx).
Status codes in the RFC 2616.

Response Example

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Hello World!</h1>
(more file contents)
  .
  .
  .
</body>
</html>
```

Some Response Codes

2xx Success:

- **200 OK** - The request has succeeded. The information returned with the response is dependent on the method used in the request.
    - **GET** an entity corresponding to the requested resource is sent in the response.
    - **HEAD** the entity-header fields corresponding to the requested resource are sent in the response without any message-body.
    - **POST** an entity describing or containing the result of the action.
- **201 Created** - The request has been fulfilled and resulted in a new resource being created.

Some Response Codes

2xx Success:

- **202 Accepted** - The request has been accepted for processing, but the processing has not been completed.
- **204 No Content** - The server has fulfilled the request but does not need to return an entity-body.
- **206 Partial Content** - The server has fulfilled the partial GET request for the resource. The request MUST have included a *Range* **header**.

Some Response Codes

3xx Redirect:

- **301 Moved Permanently** - The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs. The new permanent URI should be given by the *Location* **header** in the response.
- **304 Not Modified** - If the client has performed a conditional GET request and access is allowed, but the document has not been modified.

Some Response Codes

4xx Client Error:

- **400 Bad Request** - The request could not be understood by the server due to malformed syntax.
- **401 Unauthorized** - The request requires user authentication. The response **must** include a *WWW-Authenticate* **header** containing a challenge applicable to the requested resource.
- **403 Forbidden** - The server understood the request, but is refusing to fulfill it. Authentication will not help and the request should not be repeated.

Some Response Codes

4xx Client Error:

- **404 Not Found** - The server has not found anything matching the requested URL.
- **405 Method Not Allowed** - The method specified in the request is not allowed for the resource identified by the URI. The response must include an *Allow* **header** containing a list of valid methods.
- **408 Request Timeout** - The client did not produce a request within the time that the server was prepared to wait.

Some Response Codes

418 I'm a teapot - "Any attempt to brew coffee with a teapot should result in the error code "418 I'm a teapot". The resulting entity body MAY be short and stout." -- RFC2324

> *This error is a reference to Hyper Text Coffee Pot Control Protocol which was an April Fools' joke in 1998.*

Some Response Codes

5xx Server Error:

- **500 Internal Server Error** - The server encountered an unexpected condition that prevented it from fulfilling the request.
- **502 Bad Gateway** - The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.
- **503 Service Unavailable** - The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

All response codes

Headers

Client Headers

- **Accept** Content-Types that are acceptable for the response
  (text/html, image/jpeg, ...).
- **Accept-Charset** Character sets that are acceptable (utf-8, iso-8859-1, ...).
- **Accept-Encoding** List of acceptable encodings (gzip, deflate, ...).
- **Accept-Language** List of acceptable human languages for response.
- **Connection** What type of connection the user-agent would prefer
  (keep-alive, ...).
- **Cookie** A HTTP cookie previously sent by the server with **Set-Cookie**.
- **Content-Length** The length of the request body in octets (8-bit bytes).

Client Headers

- **Content-Type** The MIME type of the body of the request (used with POST and PUT requests).
- **Date** The date and time that the message was sent.
  Date: <day-name>, <day> <month> <year> <hour>:<minute>:<second> GMT
- **Host** The **domain name** of the server (for virtual hosting), and the TCP **port number** on which the server is listening. The port number may be omitted if the port is the standard port for the service requested. **Mandatory since HTTP/1.1**.
- **If-Modified-Since** Allows a **304 Not Modified** to be returned if the content is unchanged.
- **Range** Request only part of an entity. Bytes are numbered from 0.
- **User-Agent** The user agent string of the user agent.

Client Header Examples

```
Accept: text/plain
Accept-Charset: utf-8
Accept-Encoding: gzip, deflate
Accept-Language: en-US
Connection: keep-alive
Cookie: username=johndoe; session_id=7f3fe5016a9cda0c4adbd44aeea9d5
Content-Length: 348
Content-Type: application/x-www-form-urlencoded
Date: Tue, 15 Nov 1994 08:12:31 GMT
Host: www.google.com:80
If-Modified-Since: Sat, 29 Oct 2014 19:43:31 GMT
Range: bytes=500-999
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101
```

Server Headers

- **Accept-Ranges** What partial content range types this server supports.
- **Allow** Valid actions for a specified resource. To be used for a **405 Method not allowed**.
- **Cache-Control** Tells all caching mechanisms from server to client whether they may cache this object. It is measured in seconds. (max-age=<seconds>, no-cache)
- **Content-Encoding** The type of encoding used on the data.
- **Content-Language** The language the content is in. (pt-PT, en-US, ...)
- **Content-Length** The length of the response body in octets (8-bit bytes)

Server Headers

- **Content-Location** An alternate location for the returned data.
- **Content-Range** Where in a full body message this partial message belongs.
- **Content-Type** The MIME type of this content.
- **Expires** Gives the date/time after which the response is considered stale.
- **Last-Modified** The last modified date for the requested object.
- **Location** Used in redirection, or when a new resource has been created.
- **Set-Cookie** A HTTP cookie.

> *The Multipurpose Internet Mail Extensions (MIME) type is a standardized way to indicate the nature and format of a document.*

Example Server Headers

```
Accept-Ranges: bytes
Allow: GET, HEAD
Cache-Control: max-age=36001
Content-Encoding: gzip
Content-Language: da
Content-Length: 348
Content-Location: /index.htm
Content-Range: bytes 21010-47021/47022
Content-Type: text/html; charset=utf-8
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
Location: http://www.w3.org/pub/WWW/People.html
Set-Cookie: session_id=7f...; Domain=foo.com; Path=/; Expires=Wed,
```

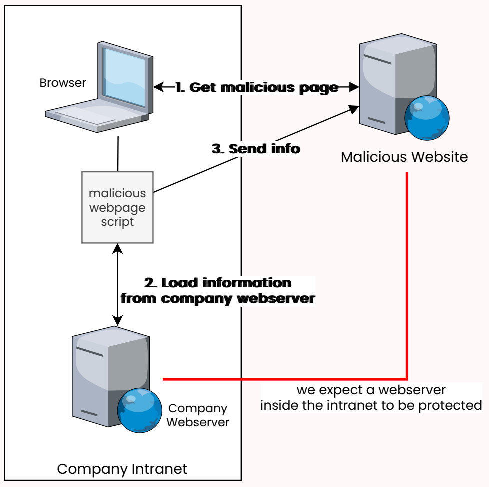
More on header fields as described in RFC2616.

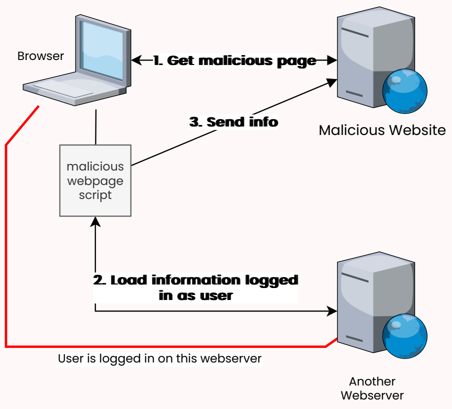
SOP and CORS

SOP: Same-origin policy

A **security mechanism** that restricts how a **document/script** loaded by **one origin** can interact with a resource from **another origin**.

Two URLs have the **same origin** if the **protocol**, **port** (if specified), and **host** are the same for both.





CORS: Cross-Origin Resource Sharing

An **HTTP-header-based mechanism** for web **servers** to indicate origins from which a browser **should allow loading** resources.

- By default, **modern browsers** follow **SOP** for **Ajax** requests. The rules about which requests should be allowed are complicated and fuzzy.
- **CORS** can be used to allow **more origins**.
- For extra security, CSRF tokens should be used. More on this later.

Browsers use CORS by doing an initial **preflight request** using the **OPTIONS** method and these headers:

- Origin: the origin that caused the request (scheme, hostname and path).
- Access-Control-Request-Method: which method will be used.
- Access-Control-Request-Headers: which headers will be sent.

Preflight Request Example

Imagine the following code running on *https://foo.org/somepage.php*.

```
async function postData(data) {
  return fetch('https://bar.com/savedata.php', {
    method: 'post',
    headers: {
      'Content-Type': 'application/x-www-form-url
    },
    body: encodeForAjax(data)
  })
}
```

This would be a possible **preflight request** sent to the server at *bar.com*:

```
OPTIONS /savedata.php HTTP/1.1
Origin: https://foo.org
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Content-Type
```

Preflight Response Example

This would be a **possible response**:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://foo.org
Access-Control-Allow-Method: POST
Access-Control-Allow-Headers: Content-Type
Access-Control-Max-Age: 86400
```

Access-Control-Allow-Origin could be * to allow requests from any origin.

As access was **granted**, the **actual request** would follow.

REST

REST Cook Book

REST

REST (**Re**presentational **S**tate **T**ransfer) is a resource-based architecture style for designing networked applications.

- **Uniform Interface**: the system is comprised of named resources accessed using a generic interface.
- **Client-Server**: separating the user interface concerns from the data storage concerns.
- **Stateless**: each request from the client to the server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- **Cacheable**: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
- **Layered System** - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.

First described by Roy T. Fielding in his PhD thesis.

Uniform Interface

- Things (**resources**) instead of **actions**.
  employee.php instead of *getemployee.php and saveemployee.php*.
- Individual resources are **identified** in requests using **URIs** as resource identifiers.
  e.g., employee.php?id=1234 or even just *employee/1234*
- When a client holds a representation of a resource, including any metadata attached, it has **enough information** to **modify** or **delete** the resource on the server.

Uniform Interface

Use the HTTP standard to describe communication.

`http://www.example.com/employee`

- **GET** to list all employees.
- **POST** creates a new employee.

`http://www.example.com/employee/1234`

- **GET** to get information about employee 1234.
- **PUT** means that you want to create/update employee 1234.
- **DELETE** means that you want to delete employee 1234.

Stateless

- Communication must be **stateless** in nature.
- Each request from the client to the server must contain **all of the necessary information** to understand the request, and cannot take advantage of any stored context on the server.
- Session **state** is therefore kept **entirely on the client**.

Cacheable

- Data within a response to a request should be implicitly or explicitly labeled as **cacheable** or **non-cacheable**.
- If a response is cacheable, then a client cache is **given the right to reuse** that response data for later, equivalent requests.

Content Negotiation

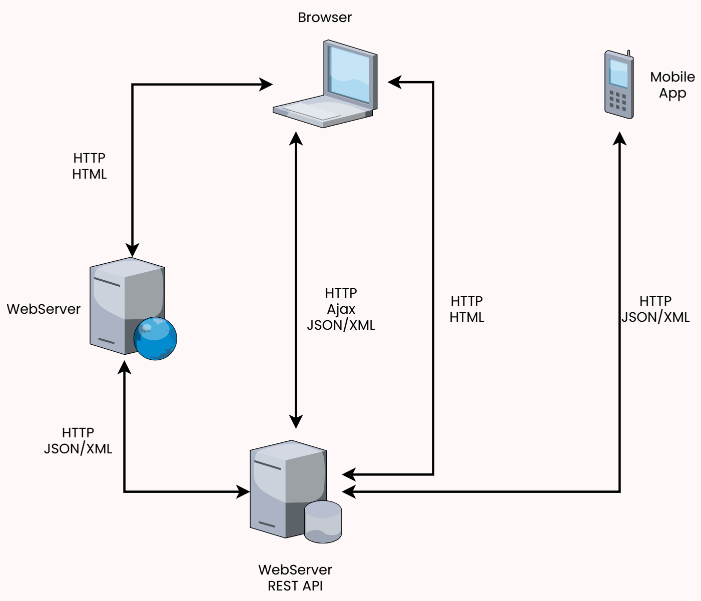
Use the **Accept** header to ask for a particular representation of the resource.

```
GET /employee/1234 HTTP/1.1
Host: www.example.com
Accept: application/json
```

```
GET /employee/1234 HTTP/1.1
Host: www.example.com
Accept: application/xml
```

```
GET /employee/1234 HTTP/1.1
Host: www.example.com
Accept: text/html
```

Scenarios


Browser
Mobile App
HTTP HTML
WebServer
HTTP Ajax JSON/XML
HTTP HTML
HTTP JSON/XML
HTTP JSON/XML
WebServer REST API

PHP and HTTP

Sending headers

To **add a header** to the response just use the **header** function:

```
header('Location: somewhere_else.php');
```

Just be careful to do it before outputting any data.

To send HTTP response codes:

```
header('HTTP/1.0 404 Nothing to see here');
```

Or:

```
http_response_code(418);
```

Finding HTTP method

To find which **HTTP method** was used to access the resource use the **$_SERVER** array:

```
if ($_SERVER['REQUEST_METHOD'] == 'PUT') {
  // update resource
}
```

Finding the Accept header

To find the **Accept** header sent by the client we can also use the **$_SERVER** array:

```
if ($_SERVER['HTTP_ACCEPT'] == 'application/json'
  echo json_encode($employees);
}
```

Other headers can also be found in the $_SERVER array or using the apache_request_headers function.

```
$headers = apache_request_headers();

foreach ($headers as $header => $value) {
    echo "$header: $value <br />\n";
}
```

CORS in PHP

Allowing from any origin:

```php
<?php
  if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
      header('Access-Control-Allow-Origin: *');
      header('Access-Control-Allow-Methods: GET')
      header('Access-Control-Allow-Headers: Conte
      header('Access-Control-Max-Age: 86400'); //
      die();
  }

  // normal request...
?>
```

CORS in PHP

Allowing from a specific origin:

```php
<?php
  if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    if ($_SERVER['HTTP_ORIGIN'] === 'http://bar.c
      header('Access-Control-Allow-Origin: http:/
      header('Access-Control-Allow-Methods: GET')
      header('Access-Control-Allow-Headers: Conte
      header('Access-Control-Max-Age: 86400'); //
    } else {
      header("HTTP/1.1 401 Unauthorized");
    }
    die();
  }

  // normal request...
?>
```