How HTTP Works

In this lesson, we'll look at the mechanics of HTTP a little more closely.

We'll cover the following

As we've seen before, HTTP follows a request/response model where a client connected to the server issues a request and the server replies back to it.

An HTTP message (either a request or a response) contains multiple parts:

- start line
- headers
- body

HTTP requests

In a request, the start line indicates the verb used by the client, the path of the resource it wants, and the version of the protocol it is going to use.

```
GET /players/lebron-james HTTP/1.1
```

In this case, the client is trying to `GET` the resource at `/players/lebron-james` through version `1.1` of the protocol. This shouldn't be nothing hard to understand.

Headers

After the start line, HTTP allows us to add metadata to the message through headers which take the form of key-value pairs separated by a colon.

```
GET /players/lebron-james HTTP/1.1
Host: nba.com
```

```
Accept: */*
Coolness: 9000
```

In this request, for example, the client has attached three additional headers to the request, `Host`, `Accept`, and `Coolness`.

Wait, `Coolness`?!?!

Headers don't have to use specific, reserved names, but it's generally recommended to rely on the ones standardized by the HTTP specification. The more you deviate from the standards, the less the other party in the exchange will understand you.

`Cache-Control` is, for example, a header used to define whether a response is cacheable. Most proxies and reverse proxies understand this as they follow the HTTP specification to the letter. If you were to rename your `Cache-Control` header to `Awesome-Cache-Control`, proxies would have no idea how to cache the response as they're not built to follow the specification you just came up with.

> **ℹ Standard headers to the rescue of your servers**
>
> The HTTP specification considers multiple scenarios and has created headers to deal with a plethora of situations. `Cache-Control` helps you scale better through caching. `Stale-If-Error` makes your website available even if there's a downtime, this is one of those headers that should be understood extremely well, as it can save a lot of troubles. `Accept` lets the client negotiate what kind of `Content-Type` is best suited for the response.
>
> For a complete list of headers I would recommend taking a look at this exhaustive Wikipedia article

Custom headers

Sometimes it might make sense to include a custom header in the message, as you might want to add metadata that is not a part of the HTTP spec. A server could choose to include technical information in its response so that the client can simultaneously execute requests and get important information regarding the status of the server that's replying back.

```
...
X-Cpu-Usage: 40%
```

```
X-Memory-Available: 1%
...
```

When using custom headers, it is always preferred to prefix them with a key so that they won't conflict with other headers that might become standard in the future. Historically, this worked until everyone started to use "non-standard" `X` prefixes which, in turn, became the norm. The `X-Forwarded-For` and `X-Forwarded-Proto` headers are examples of custom headers that are widely used and understood by load balancers and proxies, even though they weren't part of the HTTP standard.

If you need to add your own custom header, it's generally better to use a vendor prefix, like `Acme-Custom-Header` or `A-Custom-Header`.

After the headers, a request might contain a body, which is separated from the headers by a blank line.

```
POST /players/lebron-james/comments HTTP/1.1
Host: nba.com
Accept: */*
Coolness: 9000

Best Player Ever
```

Our request is now complete with a start line (location and protocol information), headers, and a body. Note that the body is completely optional, and, in most cases, is only used when we want to send data to the server. That is why the example above used the verb `POST`.

HTTP responses

A response is not very different:

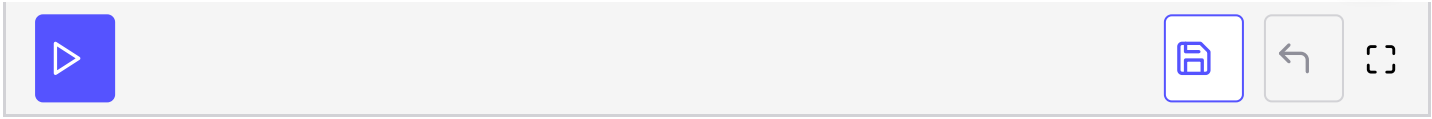
```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: private, max-age=3600

{"name": "Lebron James", "birthplace": "Akron, Ohio", ...}
```

The first information that the response advertises is the version of the protocol it uses, and the status of this response. Headers follow suit and, if required, a line

break, followed by the body.

Try receiving some live HTTP responses below! Just replace the URL with any HTTP URL you'd like.

```
curl -I -s neverssl.com
```



As mentioned, the protocol has undergone numerous revisions and has added features over time like new headers, status codes, etc., but the underlying structure hasn't changed much, i.e., start line, headers, and body.

What has changed is how the clients and servers are exchanging those messages; let's take a closer look at that in the next lesson.