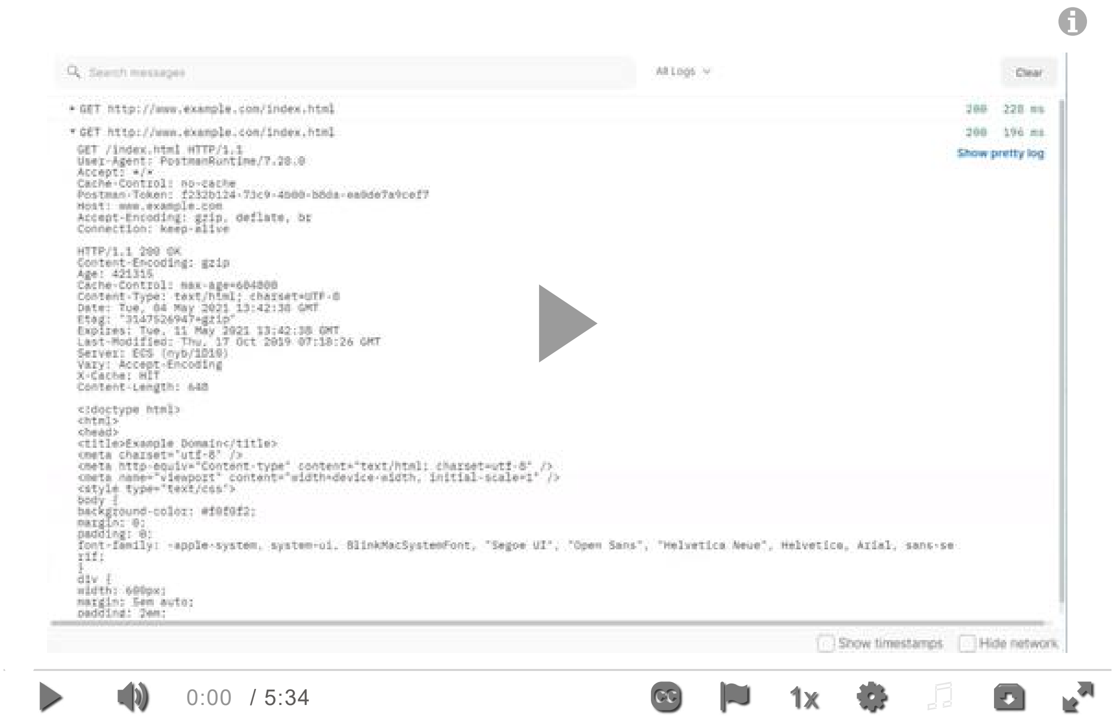
# Exploration — HTTP Part 2: The Response

## Introduction

Let us now continue our example and now focus on the HTTP response.



```
• GET http://www.example.com/index.html                    200   228 ms
▾ GET http://www.example.com/index.html                    200   196 ms
  GET /index.html HTTP/1.1
  User-Agent: PostmanRuntime/7.28.0
  Accept: */*
  Cache-Control: no-cache
  Postman-Token: f232b124-73c9-4b00-b0da-ea0de7a9cef7
  Host: www.example.com
  Accept-Encoding: gzip, deflate, br
  Connection: keep-alive

  HTTP/1.1 200 OK
  Content-Encoding: gzip
  Age: 421315
  Cache-Control: max-age=604800
  Content-Type: text/html; charset=UTF-8
  Date: Tue, 04 May 2021 13:42:38 GMT
  Etag: "3147526947+gzip"
  Expires: Tue, 11 May 2021 13:42:38 GMT
  Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
  Server: ECS (nyb/1D10)
  Vary: Accept-Encoding
  X-Cache: HIT
  Content-Length: 648

  <!doctype html>
  <html>
  <head>
  <title>Example Domain</title>
  <meta charset="utf-8" />
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style type="text/css">
  body {
  background-color: #f0f0f2;
  margin: 0;
  padding: 0;
  font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-se
  rif;
  }
  div {
  width: 600px;
  margin: 5em auto;
  padding: 2em;
```

## Generating the HTTP Response at the Web Server

At the web server the HTTP request is processed and an HTTP response is sent back to the client. The basic functionality of a web server, in a simplified form, is as follows:

1. Wait for HTTP requests.
2. Accept the request.
3. Interpret the URL in the request.
4. Either process the request itself or invoke a program installed at this web server to process the request.
5. Send back the HTTP response.
6. Go back to 1.

Typically, when the requested resource is a static HTML file or an image file, then the web server reads the file and sends it back without needing to invoke a separate server program. However, for any dynamic content, the web server will invoke a program that processes the request and generates the response. The web server will then send back this response to the client. For example, we would expect that almost all Canvas web pages are generated by invoking a program, which according to **Wikipedia** **(https://en.wikipedia.org/wiki/Instructure)** is written in the Ruby programming language. For most web applications, a server program is very likely to make requests to other programs. For example, any substantial website will use a database management system for persistent storage of data.

## HTTP Response: What is in it?

Continuing with our example, the following HTTP response is sent back for our request for **http://www.example.com/index.html** **(http://www.example.com/index.html)**.

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Age: 918
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Thu, 25 Feb 2021 12:20:08 GMT
Etag: "3147526947+gzip"
Expires: Thu, 04 Mar 2021 12:20:08 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (nyb/1D10)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 648

<!doctype html>
<html>
<head>
    <title>Example Domain</title>
    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetic
a Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>
<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

As you might have noticed, the structure of an HTTP response is very similar to that of **an HTTP request and has up to 4 parts.**

1. **Status line.**
    - This is the first line in the request and has the following format `HTTP/Version Status-code Reason-phrase`
        - In our example, its value is `HTTP/1.1 200 OK`
    - The status line starts with the version of HTTP protocol being used,
        - In our example, the version is 1.1.
    - This is followed by the **status-code** of the response which is a 3 digit integer.
        - In our example, the status code is 200.
    - This is followed by the Reason-phrase, which is a small textual description of the status code.
        - For status code 200, the Reason-phrase is `OK`.
    - We will look at response codes in greater detail later on. For now, we note that status codes in
        - 200's mean that the request was successful.
        - 400's mean that the request failed because of a client error, e.g., the request's syntax was incorrect.
        - 500's mean that the request failed because of a sever error, e.g., the server code threw an exception.
2. **Response headers.**
    - The status line is followed by response headers.
    - These headers are in the form of `name:value` pairs, just like the request headers.
    - The HTTP specification does not require any particular response headers that must be contained in every response.
    - However, practically all responses contain at least a `Date` and `Content-Type` header.
    - The `Content-Type` header tells the browser the format of the resource sent in the body of the response.
        - E.g., the `Content-Type` header in our example is `Content-Type: text/html; charset=UTF-8`
        - This means that the request is in HTML and uses the UTF-8 character set.
3. **A blank line.**
4. **An optional body.**

- The response body contains the resource.
- In our example, the body contains the complete HTML document, which a browser will display to the user.
- Even though the body part is optional, the HTTP response sent back for most HTTP methods contains a body part.
- However, there are certain methods for which there is no body.

## Processing the Response at the Web Client

The web client now receives the response. Assuming the status code is 200 (OK), the web client will use the response headers, in particular, the `Content-Type` header to determine what to do with the response body. Consider that the web client is a browser.

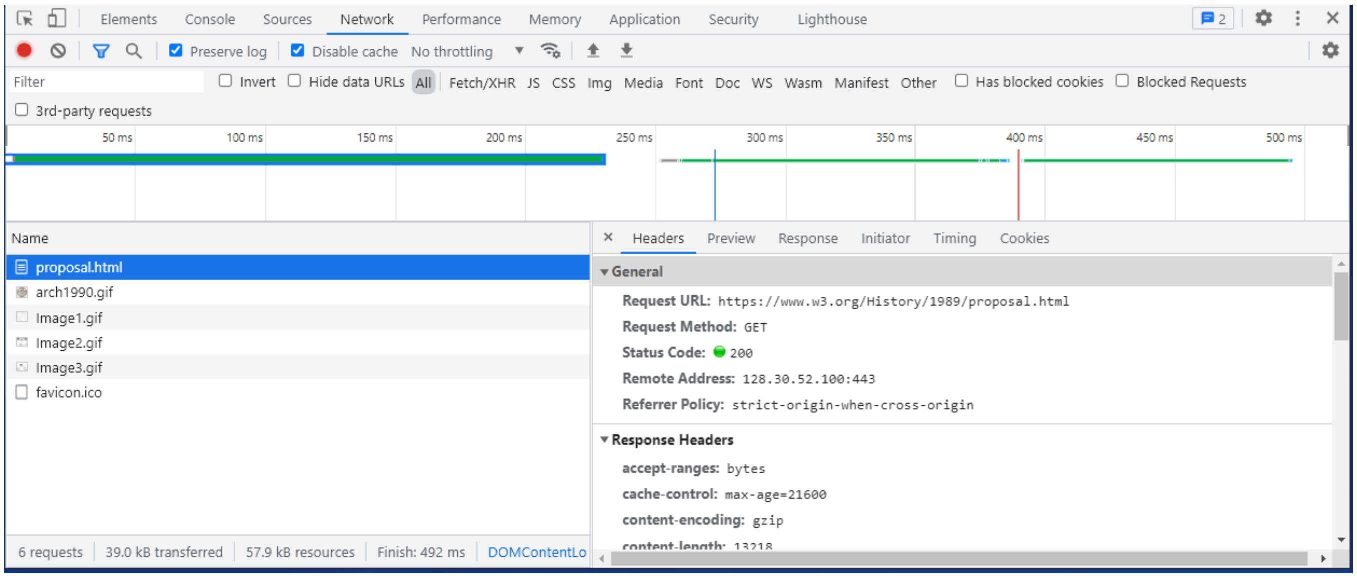
- For HTML content, the client will render the document.
- If the content type was mpeg, the browser might invoke a helper application or a plug-in to display the document.

## Exercise

Start a Chrome browser. Open up "Developer Tools" and go to the "Network" tab. There are multiple ways to open "Developer Tools" such as:

- Enter "Ctrl-Shift-I" (Windows) or ⌥-⌘-I aka Option-Command-I (Mac)
- Use the menu "More tools" -> "Developer Tools" or
- "Right-Click" -> "Inspect."

Now go to the URL **https://www.w3.org/History/1989/proposal.html (https://www.w3.org/History/1989/proposal.html)** , which is the website for the original proposal for the web project. In the box with the title "Name" click on the name of the resource that was requested, i.e., proposal.html.



Name
proposal.html
arch1990.gif
Image1.gif
Image2.gif
Image3.gif
favicon.ico

6 requests | 39.0 kB transferred | 57.9 kB resources | Finish: 492 ms | DOMContentLo

▼ General

```
Request URL: https://www.w3.org/History/1989/proposal.html
Request Method: GET
Status Code: ● 200
Remote Address: 128.30.52.100:443
Referrer Policy: strict-origin-when-cross-origin
```

▼ Response Headers

```
accept-ranges: bytes
cache-control: max-age=21600
content-encoding: gzip
content-length: 13218
```

- Examine the HTTP request and the HTTP response.
- For this URL which headers are specified in the request and response that also appear in the URL **http://www.example.com/index.html (http://www.example.com/index.html)** that we looked at earlier in this exploration.
- If you compare these common headers, for which headers are the value the same and for which headers are the values different across the 2 URLs?

## Summary

In this and the previous exploration, we studied an HTTP request and response from end-to-end. We highlighted the basic structures of HTTP requests and HTTP responses. We are going to continue to deepen our understanding of the HTTP protocol in subsequent modules.

## Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- The specifications of different web technologies are available as Request-for-Comments or **RFCs (https://en.wikipedia.org/wiki/Request_for_Comments)**. Here are the RFC for some HTTP versions: **HTTP 1.0 (https://tools.ietf.org/html/rfc1945)**, **HTTP 1.1 (https://tools.ietf.org/html/rfc2616)**
- Mozilla Developer Network or **MDN (https://developer.mozilla.org/en-US/)** is an excellent reference for all types of information about web development. A lot of information about the HTTP protocol (e.g., response codes, request and response headers) is available at MDN in a more user-friendly form than the formal RFCs.