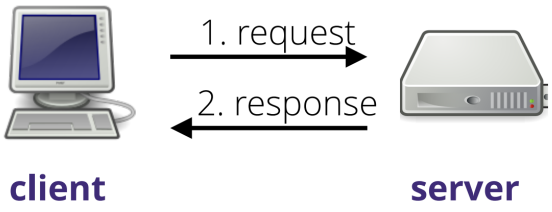
Lecture Notes: HTTP

▼ **Definitions:**

- **HTTP:** HyperText Transfer Protocol
- **RFC:** Requests for Comments
- **Blocking Call:** The server waits for a message to arrive in the form of a request
- **Response Codes:** Three digit numbers that provide a way of quickly explaining the message from the server.

Protocol

- A simple Protocol is the Client-Server Protocol, where the Client sends a request and receives a response back. e.g. A client could be a computer communicating with an external hard drive and ask for memory space in block 100.



- The response from the client would be the address of the memory space and something to indicate that it was a successful request ping.

Server

- The server code is a continuous while loop that will look for requests from the client.

Here is a conceptual outline of a server code:

1. The server reads a request from a socket it is listening on (blocking call
2. Once the message has arrived, the server will serve will request. E.g. A web server the request will be to access a specific web page, the server will complete this request by reading files from its file system
3. The server will send back a response

```
Server()
 while (1) {
 req = readreq();
 response = serve(req);
 write(resp);
 }
```

HTTP Requests:

`GET /index.html HTTP/1.1`

1. GET: This is the method that we are preforming for this particular HTTP request. `GET` allows us to get a resource from the server. In the case below we are getting a file specified by the path: "/index.html". Finally we also indicate what version of HTTP we are using, 1.1 (universally accepted)

`HOST: www.bristol.ac.uk`

2. We provide the host we think we are connected to, in this case it is "www.bristol.ac.uk"

`Connection: Close`

3. Then we provide what we want the server to do with our request, here, once the request has been completed the connection will close immediately.

HTTP Request

```
GET /index.html HTTP/1.1
Host: www.bristol.ac.uk
Connection: close
```

Methods used in HTTP:

- **GET:** gets a resource (like webpage)
- **POST:** posting a data to the server, which the server should know how to handle, e.g. uploading a file to a server
- **HEAD:** This will only ask for the headers of the response. e.g. if you wan to download a file, but first you want to check how long that file is. So you might first send a HEAD request.
- **PUT:** is a version of post, where in POST you might be sending a list of uploads. PUT will replace a upload from that list

The most common methods that are used and implemented are GET and POST

HTTP Responses:

`HTTP/1.1 200 OK`

1. This tells you the version of HTTP being used in the request. 200 OK is a **response code**; a way of declaring a successful response.

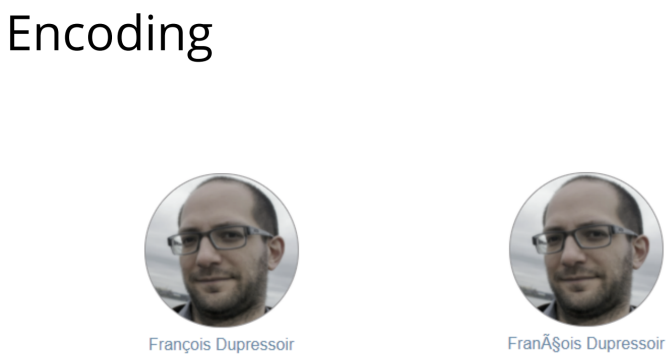
Headers:

`Content-Type: text/html; charset=UTF-8`

`Content-Length`

1. What is the content type of the the response, in this case it is a text/html file. Charset is used when the content type is of text, this explains which form of

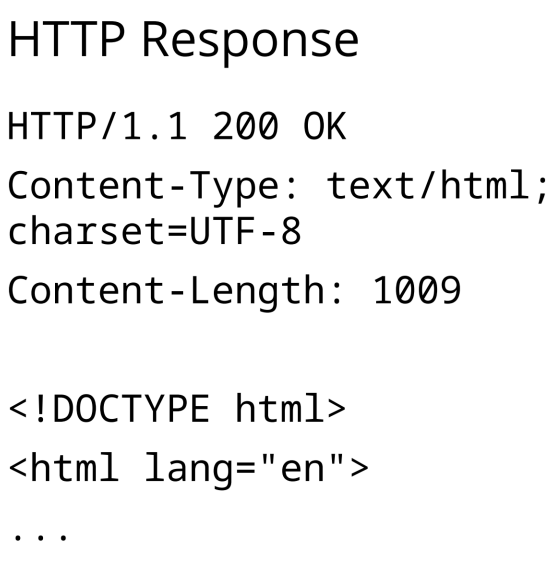
text encoding is being used in this document. Wrong encoding can lead to error such as these:



2. The length of the content

Main body of the response:

- In this case is a HTML document, reflecting that we requested a web page



Response Codes:

- 1XX: Information Codes; If the server has received a request and is still working on the request sent.

- 2XX: success; (200 OK, 201 Created, ...), here for example 201 means a resource was successfully created, meaning your request caused the server to create a resource
- 3XX: redirect; (301 Moved Permanently...) this means that the resource you have requested has been moved elsewhere, (different webpage for example).
- 4XX: client error; (400 Bad Request, 403 Forbidden, 404 Not Found,....)
- 5XX: server error; (500 Internal Server Error,...)

Content-Type:

This explains what the content type of the body will be, for web pages this is text-html files.

There exists many different type of content headers that describe the different types of contents:

Examples:

- text / plain
- text / html
- image / jpeg
- application / pdf
- video / mp4

It is important for the client i.e. the web browser to know how to handle the response. e.g. you want the web browser to parse a video as a video and not a HTML file.

Example: Sending a request to server.

- Here we are using wget to send the request
- The output is directed to the standard output and standard error
- And we are only printing the headers of the response

```
wget -S -O - bristol.ac.uk 2>&1 | head -31
#Explanation of Code:
-S : # Print the response headers
-O: # the output is directed to standard output and standard er
```

▼ **Output / Comment Explanations:**

```
--2024-04-14 18:04:29--  http://bristol.ac.uk/
Resolving bristol.ac.uk (bristol.ac.uk)... 137.222.1.237 ## I
Connecting to bristol.ac.uk (bristol.ac.uk)|137.222.1.237|:80
# for HTPP to run on
HTTP request sent, awaiting response...
####### HTTP RESPONSE: ############
# Becasue we didn't specify a specific web page, our request
  HTTP/1.1 302 Moved Temporarily # The response we got was th
  Location: https://bristol.ac.uk/
  Server: BigIP
  Content-Length: 0
Location: https://bristol.ac.uk/ [following] # This is the we
--2024-04-14 18:04:29--  https://bristol.ac.uk/
Connecting to bristol.ac.uk (bristol.ac.uk)|137.222.1.237|:44
# for TLS connection
HTTP request sent, awaiting response...
  HTTP/1.1 200 OK # The response was again for HTTP/1.1 and t
  Date: Sun, 14 Apr 2024 17:04:29 GMT
  Server: Apache
  Last-Modified: Thu, 11 Apr 2024 11:15:54 GMT
  ETag: "15c06-615d04757711f"
  Accept-Ranges: bytes
  Content-Length: 89094 # Length in bytes of the document tha
  Content-Type: text/html; charset=utf-8 # Content type
  Set-Cookie: BIGipServerWWWUOB_TCP_80_POP_POOL=1426266540.20
  Keep-Alive: timeout=5, max=256
  Connection: Keep-Alive
  Set-Cookie: BIGipServerWEB-DC2-UOB-TCP443-POOL=2064259756.4
```

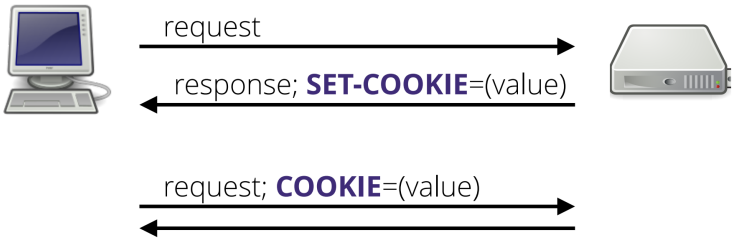
```
Length: 89094 (87K) [text/html]
Saving to: ‘index.html’

     0K ......... ......... ......... ......... ........
    50K ......... ......... ......... .......

2024-04-14 18:04:29 (980 KB/s) - ‘index.html’ saved [89094/89
```

Cookies 🍪 :

- HTTP is a stateless protocol, meaning that no memory is attained to of the request made by the client. The response is sent and then the client is forgotten. This is useful for scaling because no memory is allocated to preserving the state of the client i.e the the request that was made

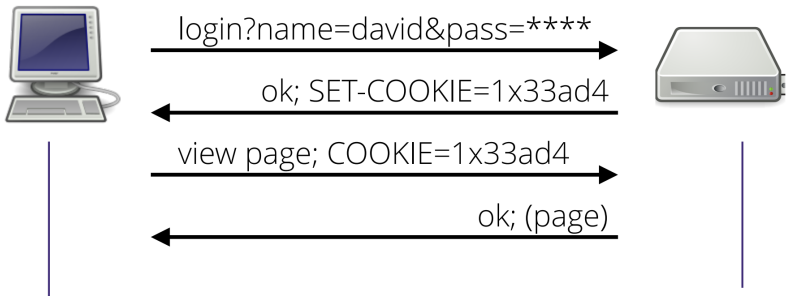
- However, as the requests that clients made were becoming more complicated i.e. we were loading web pages for more than just to read the contents of a HTML file, the protocols adapted to add **Cookies.** This allows the server to hold memory of the client and their request.
    - In the cookie protocol, a request is sent, followed by a response (like 200 OK) but also `SET-COOKIE=(value)` you can see that in the output above when we accessed bristol.ac.uk website.
    - Then when the client wants to send the same request again in the future (let me access this web page), the request will have the cookie value

Cookie protocol



- The most common usage of this is when we log into a website. Imagine you want to log into the university of bristols website as a student.
- To access any webpages within the file system i.e. the course web page / blackboard you'd have to be logged in.
- Rather than making you log in every single time you want to access a web page, the client will set a cookie, which the server will recognise and know that when that cookie is set, the user is able to access the allowed web pages.
- This is normally called a **session** where after some time the session will expire and the user will have to log in again.

Sessions



server	cookie
bristol	1x33ad4
...	...

sessions	
user	cookie
david	1x33ad4
...	...

- External cookies can also be set by external websites that allow access to specific web pages e.g. Google allowing access to the university of Bristols

website, these are known as third party cookies. These websites set their own cookies (unique identifiers) that identify your browser and track the different websites you have visited.