Web Protocols, forms and the Common Gateway Interface

Web Protocols

URLs

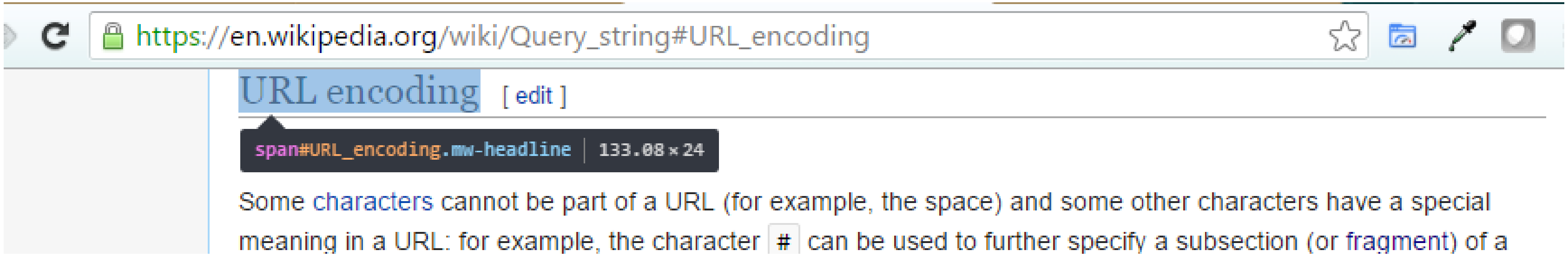
- Uniform Resource Locator
- http://www.farming-simulator.com/index.html
- Protocol :// Host Path
- The host is the unique address of the server we are accessing (this will be converted to an IP address by a DNS server)
- The path is (sometimes) the file we want to access

Query Parameters (or GET parameters)

- https://www.youtube.com/watch?v=dQw4w9WgXcQ
- Data is encoded as an add-on to a URL
- ?name=value
- Think about URL encoding (https://en.wikipedia.org/wiki/Query_string) some characters have special meaning in URLS
- https://www.youtube.com/watch?v=dQw4w9WgXcQ&t=30s
- Multiple pairs are separated by &
- This essentially allows you to turn a web request into a function
- GET parameters are directly readable in the URL bar
- GET parameters **are** sent to the server as part of the request

Fragments

- https://en.wikipedia.org/wiki/Fragment_identifier#Examples
- Fragments are **not** sent to the server
- Commonly used to scroll a page to a labelled element

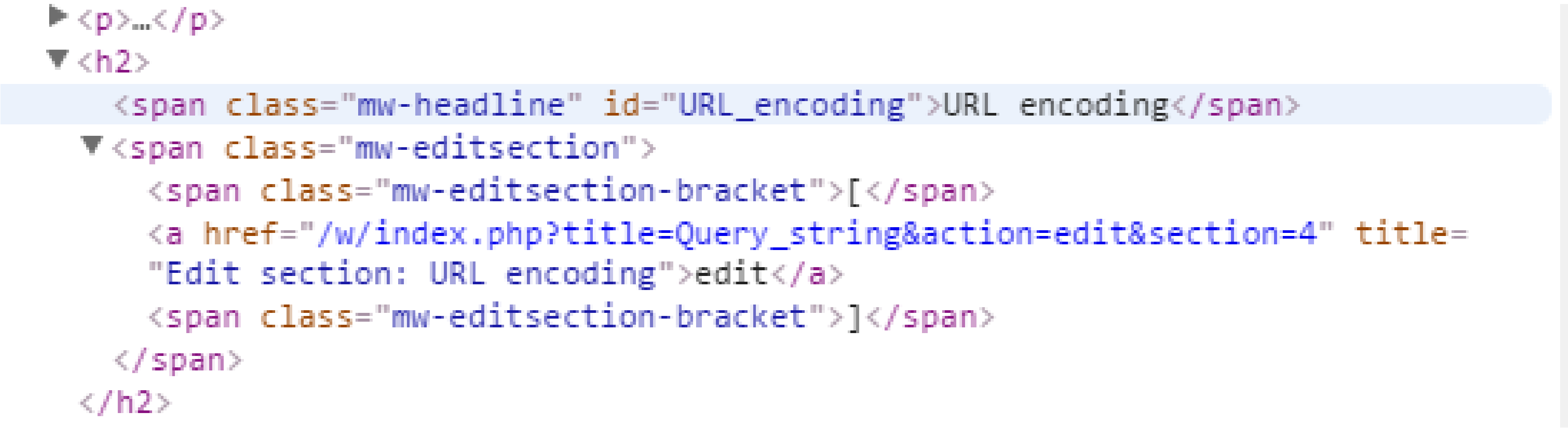


https://en.wikipedia.org/wiki/Query_string#URL_encoding

URL encoding [edit]

span#URL_encoding.mw-headline | 133.08 × 24

Some characters cannot be part of a URL (for example, the space) and some other characters have a special meaning in a URL: for example, the character # can be used to further specify a subsection (or fragment) of a



```
▶<p>…</p>
▼<h2>
   <span class="mw-headline" id="URL_encoding">URL encoding</span>
  ▼<span class="mw-editsection">
     <span class="mw-editsection-bracket">[</span>
     <a href="/w/index.php?title=Query_string&action=edit&section=4" title=
     "Edit section: URL encoding">edit</a>
     <span class="mw-editsection-bracket">]</span>
   </span>
 </h2>
```

Ports

- http://localhost:8000/
- To establish a connection you need an address and a port
- Default port is 80
- You will see the port a lot when you are testing things on a local machine since you won’t be connecting to port 80.

HTTP Request

- Hyper Text Transfer Protocol
- The request from a browser for a URL begins with a request line
- Example – we enter https://www.w3.org/Protocols/ into a URL bar
- Request line (the first line of the request) – GET /Protocols HTTP/1.1
- Method – the type of request you are making
- Path – the path we are requesting at the host
- The version of HTTP we are using
- The host is not in the request line because we connect to it before we make the request

HTTP Request Headers

- The full request is make up of the request line followed by a list of headers in Name: value pairs
- Host is useful because web servers host multiple websites – remember we are already connected to the correct machine before the request is made
- User agents are really important to give information to the server you are accessing
  - Might not be browsers, e.g. google crawler
  - When writing software which interacts with servers give an accurate user agent

```
GET /Protocols HTTP/1.1
Host: www.w3.org
User-agent: chrome v.17
```

HTTP Response

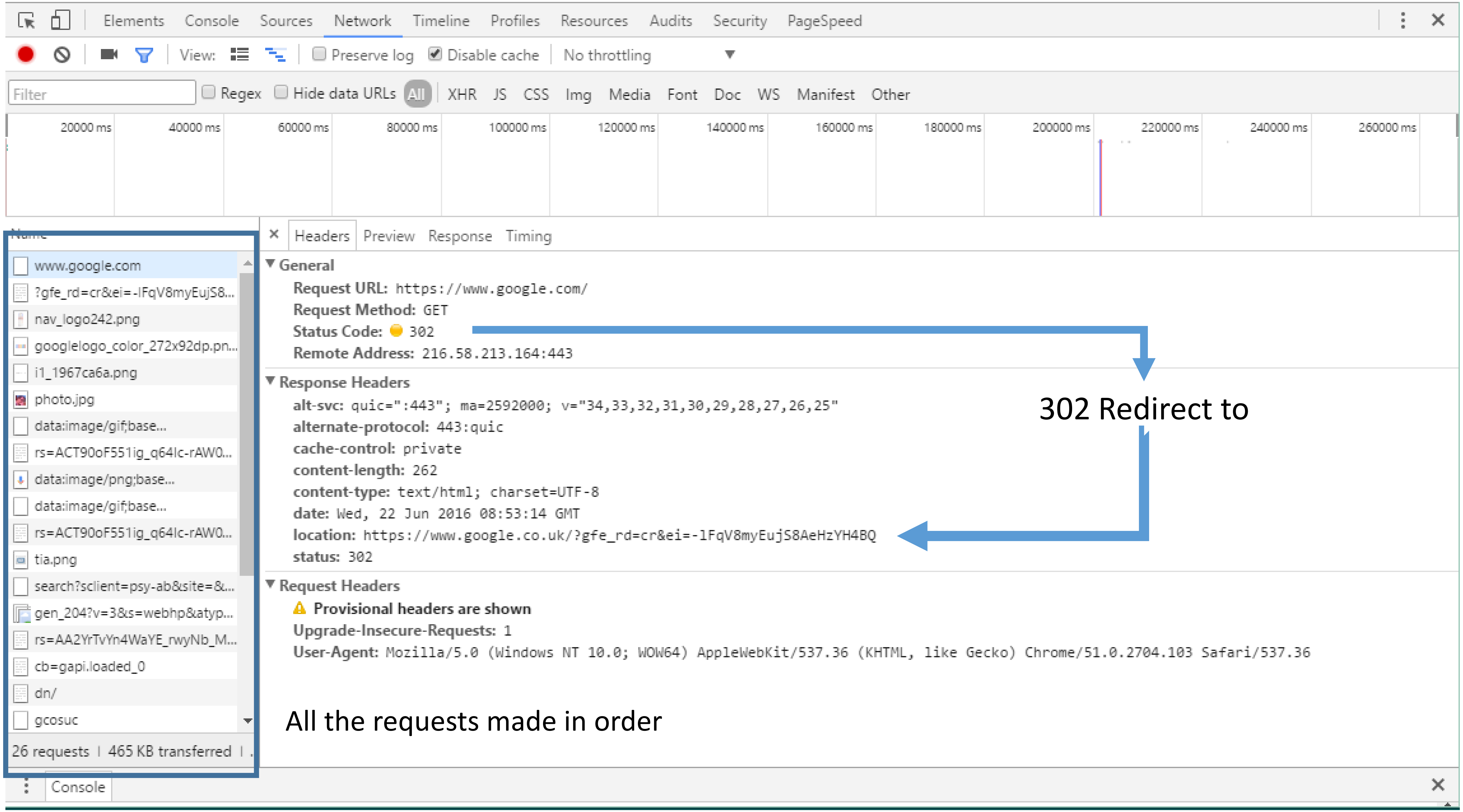
- The response from the sever begins with a status line
- Example – HTTP/1.1 200 OK
- Version
- Status code
  - 1xx – information that things are still going on, usually not used by browsers
  - 2xx – success of some kind, but not all look like success i.e. 204 is ‘no content’
  - 3xx – Redirection, either things have moved or a proxy server is involved but it means the client needs to do more
  - 4xx – Client error (e.g. 404 not found) – you screwed up
  - 5xx – Server error – I screwed up
- Reason Phrase – English language description of the status code

HTTP Response Headers

- Just like with requests these are name value pairs
- Don’t give away too much information, for example giving server software version information is just helping an attacker
- This would then be followed by the content

```
HTTP/1.1 200 OK
Date: Mon June 2016 15:49:22 GMT
Server: Apache /2.2.3
Content-Type: text/html
Content-Length: 1346
```

More chrome development tools



□ Preserve log ☑ Disable cache No throttling

□ Regex □ Hide data URLs All

www.google.com
?gfe_rd=cr&ei=-lFqV8myEujS8...
nav_logo242.png
googlelogo_color_272x92dp.pn...
i1_1967ca6a.png
photo.jpg
data:image/gif;base...
rs=ACT90oF551ig_q64lc-rAW0...
data:image/png;base...
data:image/gif;base...
rs=ACT90oF551ig_q64lc-rAW0...
tia.png
search?sclient=psy-ab&site=&...
gen_204?v=3&s=webhp&atyp...
rs=AA2YrTvYn4WaYE_rwyNb_M...
cb=gapi.loaded_0
dn/
gcosuc

26 requests | 465 KB transferred | .

General

Request URL: https://www.google.com/
Request Method: GET
Status Code: ● 302
Remote Address: 216.58.213.164:443

Response Headers

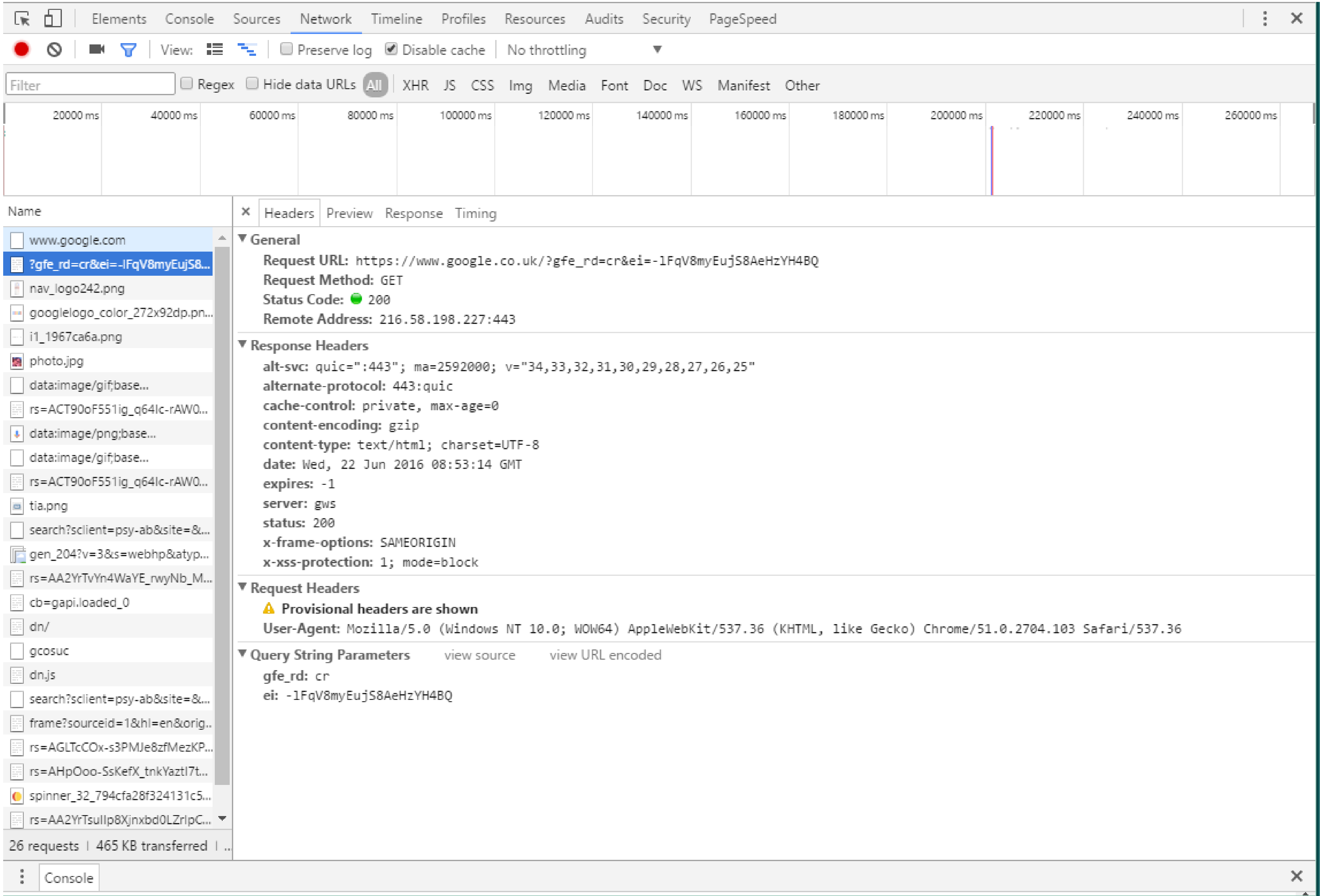
alt-svc: quic=":443"; ma=2592000; v="34,33,32,31,30,29,28,27,26,25"
alternate-protocol: 443:quic
cache-control: private
content-length: 262
content-type: text/html; charset=UTF-8
date: Wed, 22 Jun 2016 08:53:14 GMT
location: https://www.google.co.uk/?gfe_rd=cr&ei=-lFqV8myEujS8AeHzYH4BQ
status: 302

Request Headers

⚠ **Provisional headers are shown**
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36

302 Redirect to

All the requests made in order



Name

```
www.google.com
?gfe_rd=cr&ei=-lFqV8myEujS8...
nav_logo242.png
googlelogo_color_272x92dp.pn...
i1_1967ca6a.png
photo.jpg
data:image/gif;base...
rs=ACT90oF551ig_q64lc-rAW0...
data:image/png;base...
data:image/gif;base...
rs=ACT90oF551ig_q64lc-rAW0...
tia.png
search?sclient=psy-ab&site=&...
gen_204?v=3&s=webhp&atyp...
rs=AA2YrTvYn4WaYE_rwyNb_M...
cb=gapi.loaded_0
dn/
gcosuc
dn.js
search?sclient=psy-ab&site=&...
frame?sourceid=1&hl=en&orig...
rs=AGLTcCOx-s3PMJe8zfMezKP...
rs=AHpOoo-SsKefX_tnkYaztl7t...
spinner_32_794cfa28f324131c5...
rs=AA2YrTsullp8XjnxbdOLZrlpC...
26 requests | 465 KB transferred | ...
```

▼ General

```
Request URL: https://www.google.co.uk/?gfe_rd=cr&ei=-lFqV8myEujS8AeHzYH4BQ
Request Method: GET
Status Code: ● 200
Remote Address: 216.58.198.227:443
```

▼ Response Headers

```
alt-svc: quic=":443"; ma=2592000; v="34,33,32,31,30,29,28,27,26,25"
alternate-protocol: 443:quic
cache-control: private, max-age=0
content-encoding: gzip
content-type: text/html; charset=UTF-8
date: Wed, 22 Jun 2016 08:53:14 GMT
expires: -1
server: gws
status: 200
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
```

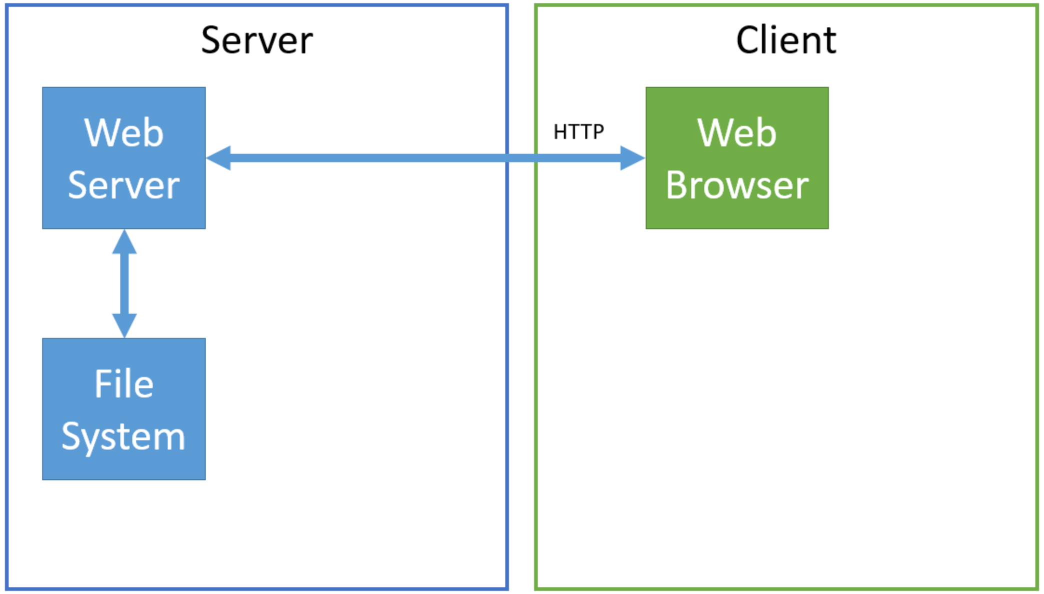
▼ Request Headers

```
⚠ Provisional headers are shown
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36
```

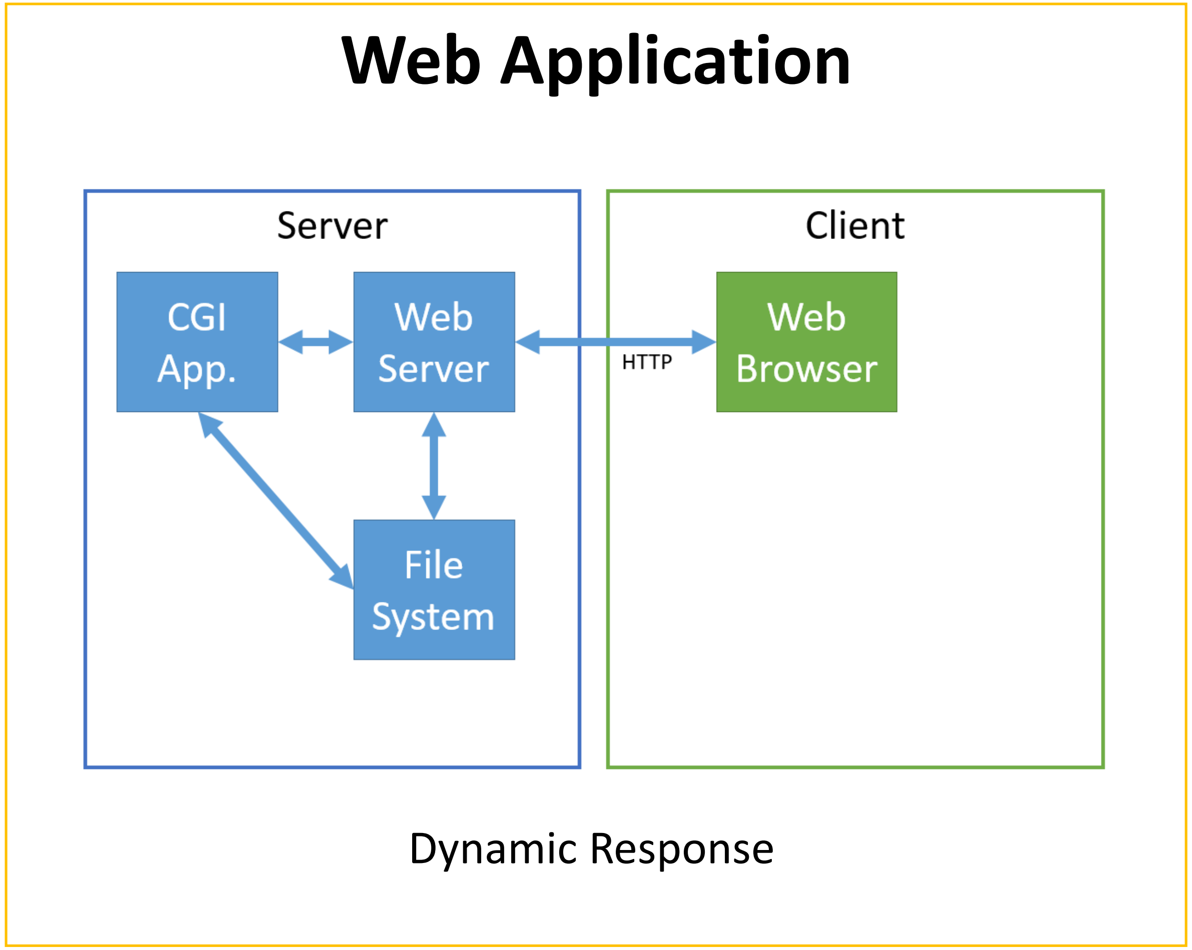
▼ Query String Parameters view source view URL encoded

```
gfe_rd: cr
ei: -lFqV8myEujS8AeHzYH4BQ
```

Types of Server Response



Static Response



Dynamic Response

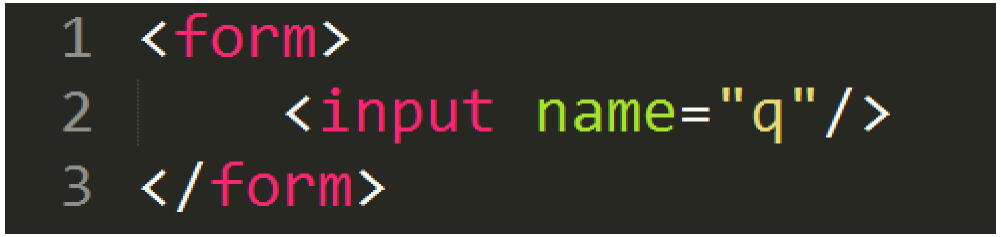
Forms

A mechanism for collecting information from a browser (or the user) and sending it to the server.

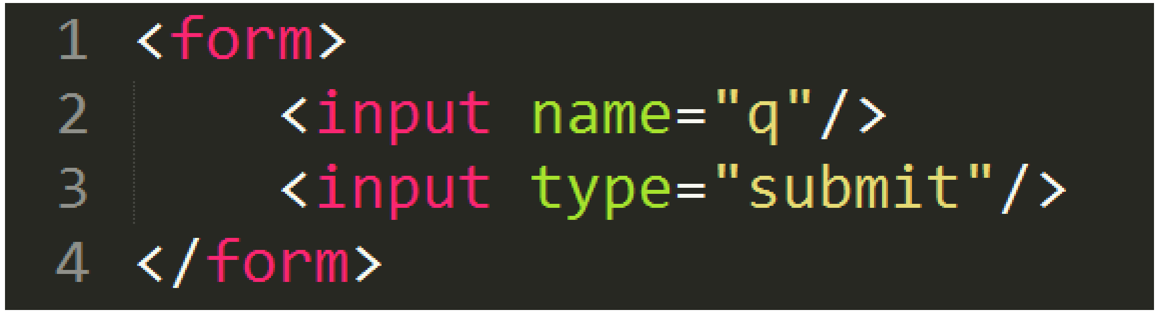
https://www.youtube.com/watch?v=2gxLcIUFs2U

HTML Forms

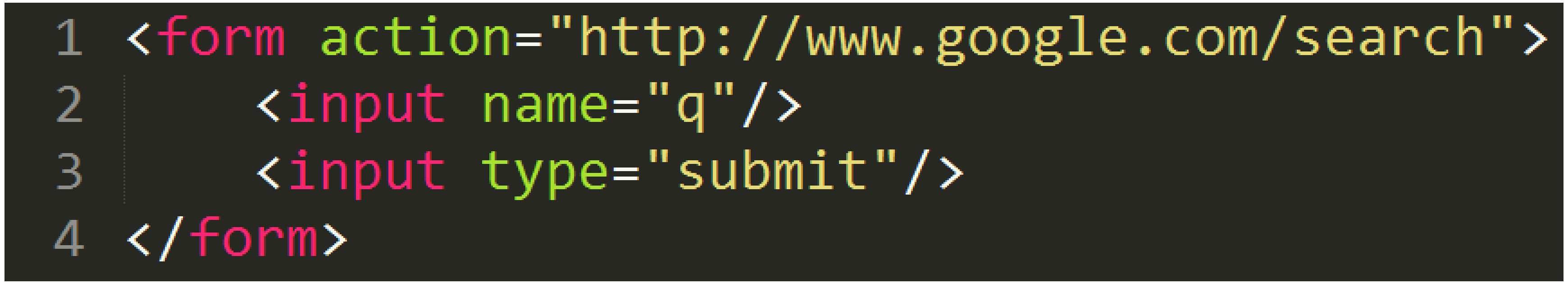
- To test forms make new html files with a text editor and open these with a browser
- Forms start and end with the <form> tags and contain form elements
- There are many different form elements and types http://www.w3schools.com/html/html_form_elements.asp http://www.w3schools.com/html/html_form_input_types.asp



```
<form>
    <input name="q"/>
</form>
```

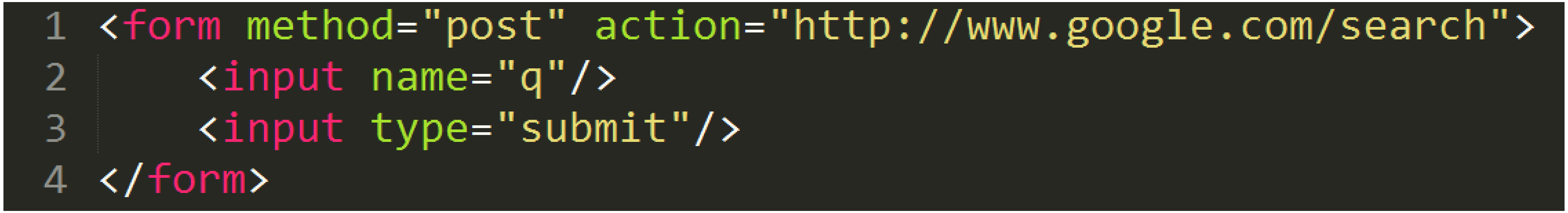


```html
<form>
    <input name="q"/>
    <input type="submit"/>
</form>
```

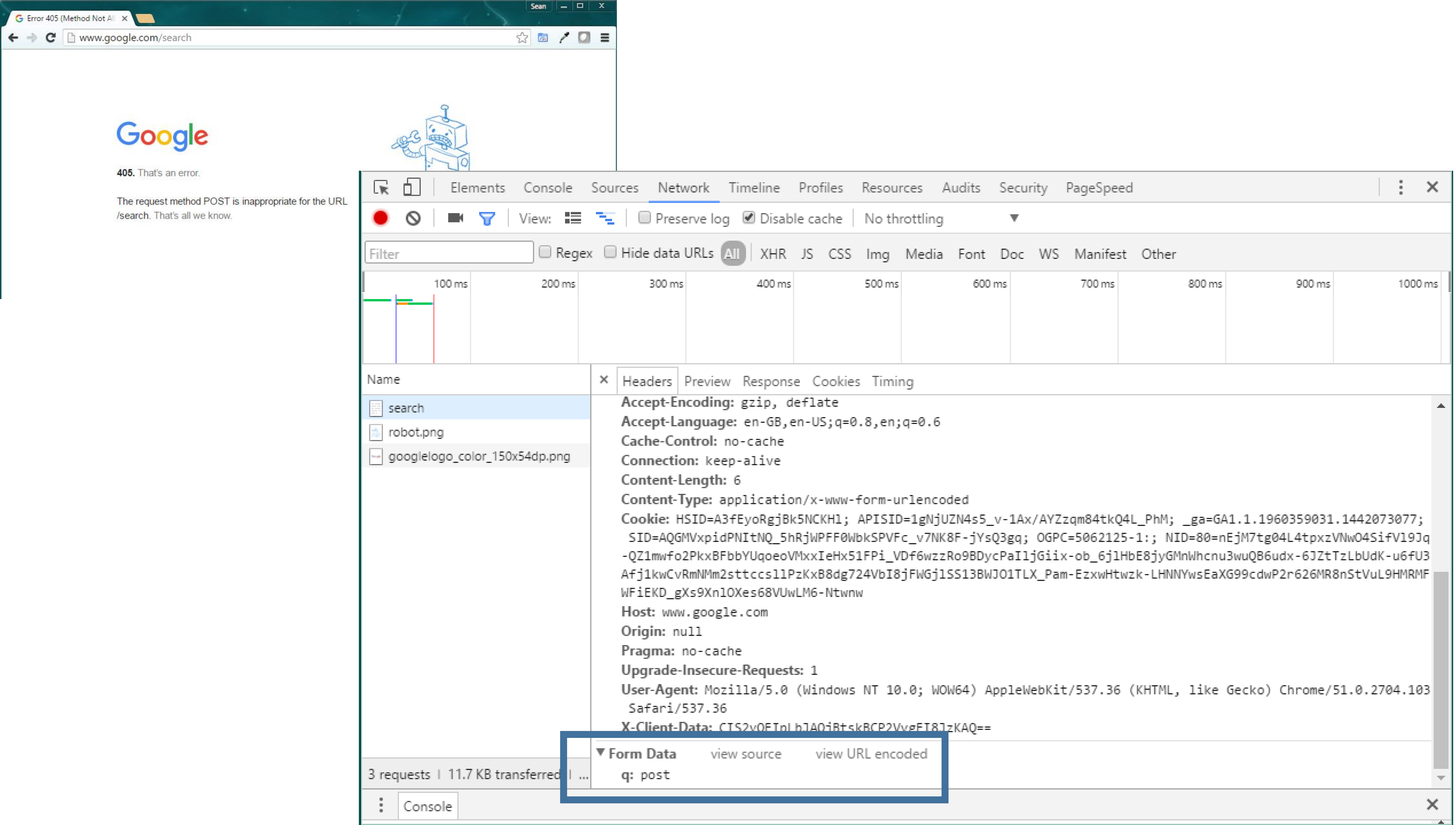


```html
<form action="http://www.google.com/search">
    <input name="q"/>
    <input type="submit"/>
</form>
```

- The action attribute specified where to send the form data to
- Using this form we can investigate URL encoding, type different strings into the text box and see how it is encoded in the URL when you hit submit… and try the same string in different browsers



```html
<form method="post" action="http://www.google.com/search">
    <input name="q"/>
    <input type="submit"/>
</form>
```



Google

405. That's an error.

The request method POST is inappropriate for the URL /search. That's all we know.

search
robot.png
googlelogo_color_150x54dp.png

3 requests | 11.7 KB transferred | ...

```
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 6
Content-Type: application/x-www-form-urlencoded
Cookie: HSID=A3fEyoRgjBk5NCKHl; APISID=1gNjUZN4s5_v-1Ax/AYZzqm84tkQ4L_PhM; _ga=GA1.1.1960359031.1442073077;
 SID=AQGMVxpidPNItNQ_5hRjWPFF0WbkSPVFc_v7NK8F-jYsQ3gq; OGPC=5062125-1:; NID=80=nEjM7tg04L4tpxzVNwO4SifV19Jq
-QZ1mwfo2PkxBFbbYUqoeoVMxxIeHx51FPi_VDf6wzzRo9BDycPaIljGiix-ob_6jlHbE8jyGMnWhcnu3wuQB6udx-6JZtTzLbUdK-u6fU3
Afj1kwCvRmNMm2sttccsllPzKxB8dg724VbI8jFWGjlSS13BWJO1TLX_Pam-EzxwHtwzk-LHNNYwsEaXG99cdwP2r626MR8nStVuL9HMRMF
WFiEKD_gXs9XnlOXes68VUwLM6-Ntwnw
Host: www.google.com
Origin: null
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103
 Safari/537.36
X-Client-Data: CIS2yQEIpLbJAQiBtskBCP2VygEI8JzKAQ==
```

▼ Form Data   view source   view URL encoded

```
q: post
```

Form methods

GET (default)

- Parameters and data is included in the URL
- Often used for fetching documents (getting)
- Limited by maximum URL length (browser variable)
- Okay to cache
- Should not change the server
- Simple fetching parameters (i.e. video id on youtube)

POST

- Parameters and data are in the body of the HTTP request after the headers
- Often used for updating data (posting)
- No maximum length
- Not okay to cache
- Okay to change the server
- Used for making server updates

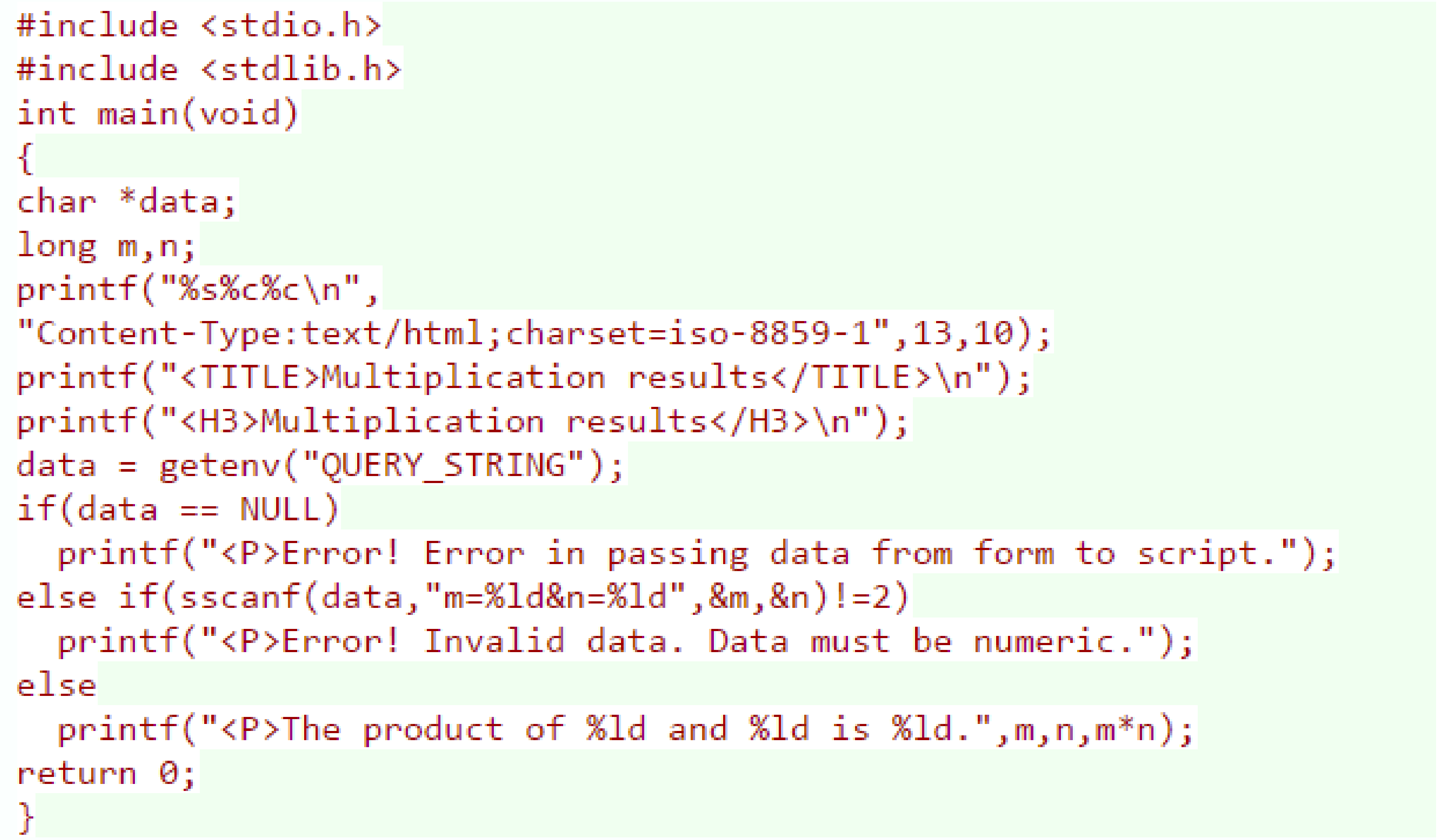
Common Gateway Interface

Common Gateway Interface (CGI) is a standard way for web servers to interface with executable programs installed on a server that generate web pages dynamically. Such programs are known as CGI scripts or simply CGIs; they are usually written in a scripting language, but can be written in any programming language.

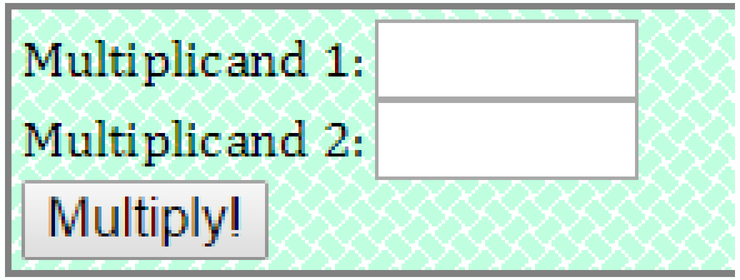
CGI Programs and Scripts

- This was essentially a ‘hack’ rather than something designed for the job
- Executable code on a web server which can be written in any programming language
- Interprets name value pairs and processes them to generate output which the client web browser can interpret (i.e. HTML)
- Data supplied over standard input for POST and via an environment variable QUERY_STRING for GET
- Various other environment variables are also passed to the executable
- Output is written to standard output and automatically sent to the browser

CGI Program example



```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
char *data;
long m,n;
printf("%s%c%c\n",
"Content-Type:text/html;charset=iso-8859-1",13,10);
printf("<TITLE>Multiplication results</TITLE>\n");
printf("<H3>Multiplication results</H3>\n");
data = getenv("QUERY_STRING");
if(data == NULL)
  printf("<P>Error! Error in passing data from form to script.");
else if(sscanf(data,"m=%ld&n=%ld",&m,&n)!=2)
  printf("<P>Error! Invalid data. Data must be numeric.");
else
  printf("<P>The product of %ld and %ld is %ld.",m,n,m*n);
return 0;
}
```



The Problems with CGI

- Security
  - Client side no real concerns
  - Server side CGI is a significant risk since they are programs run on your server
  - Clients can subvert servers to give them access to unauthorised data or to damage data
  - For this reason CGI programs usually reside in a cgi-bin directory with restricted permissions
  - CGI programmers sometimes forget to add checks to input to avoid these risks
  - CGI scripts run independently of the actual web server so you can not use the web server to police them – this isn't the case with new technologies
- Efficiency
  - Each activation of a script is a new process on the server (think about scalability)
- Engineering
  - Code managing appearance is mixed with the code managing the logic
  - Lots of repeated code which outputs the same HTML

CGI has been replaced with new purpose-built technologies, which is what we shall be looking at moving forward…