Pràctica 1: Protocol HTTP

Introducció

L'objectiu d'aquesta pràctica és l'estudi del funcionament del protocol petició-resposta HTTP. S'han de tenir clars els elements principals que intervenen en les comunicacions web i els protocols TCP/IP i HTTP)

A) Busca informació i respon les següents preguntes indicant les referències utilitzades:

1. **A quin port es reben normalment les peticions del protocol HTTP? A quina capa de la torre OSI es troba el protocol HTTP? I els protocols TCP, UDP, i IP?**
   a) Pel port tcp/80
   b) El protocol HTTP es troba a la capa o nivell 7, que corresponen a la capa d'aplicació
   c) Els protocols TCP i UDP es troben a la capa o nivell 4 o capa de transport. Una de les principals funcions d'aquest capa és la multiplexació de connexions per mitjà dels ports TCP i UDP. El protocol IP es troba a la capa o nivell 3 o capa de xarxa. Una de les principal funciones del protocol és la d'identificar equips per mitjà de la seva adreça IP.

2. **Respon les següents preguntes:**
   a) **El protocol HTTP és un protocol client-servidor.**
      VERITAT. L'afirmació és verdadera ja que el protocol HTTP segueix el model general de peticions i respostes sense estats entre un client i un servidor que opera intercanviant missatge HTTP gràcies a una connexió confiable TCP. Per tant, el funcionament consisteix en una màquina client que emet una petició dirigida a un servidor per accedir a un recurs. Aquest servidor, si té accés a aquest recurs, genera una resposta que és enviada al client. El sistema no té memòria de les peticions realitzades i cada cop que s'inicia una de nova, encara que sigui idèntica a una més antiga realitzada prèviament.
   b) **A quin camp de la capçalera d'un missatge HTTP pots trobar l'adreça IP del servidor al qual va dirigida una petició?.**
      L'adreça IP o el nom complet del servidor al qua es dirigeix una petició es por trobar a camp Host de les capçaleres d'un missatge de petició.
   c) **A quin camp de la capçalera d'un missatge HTTP pots trobar informació sobre el programa client que ha realitzat una petició?.**
      L'adreça IP del servidor al qua es dirigeix una petició es por trobar a camp User-Agent de les capçaleres d'un missatge de petició.
   d) **Què és el Request Payload Body d'un missatge HTTP?**
      En el camp de les telecomunicacions i informàtica "Payload" és la part de les dades enviades o rebudes que conté realment informació, o sigui, el missatge que volem enviar o rebre. les dades que volem rebre o enviar, i per tant no s'inclou les capçaleres i metadades necessàries per poder fer que el missatge es pugui lliurar amb èxit al receptor però que no són realment informació.
      El "Request Payload Body" és per tant la part d'un missatge de petició que hauria d'utilitzar un mètode POST o PUT (veure pregunta 5) que conté les dades que es volen enviar. Les dades es poden ser texte, imatges,etc... i poden estar codificades (que no vol dir xifrades).
   e) **Què significa que HTTP és un protocol sense estat (stateless)?**
      És un protocol que tracta cada petició de manera independent i sense cap relació amb les anteriors de manera que la comunicació consisteix d'un conjunt de parelles petició-resposta cadascuna de les quals és independent i a on el resultat de les anterior parelles petici-resposta no tenen cap influència en la actual.

3. **Indica de quina manera el client faria una petició de la pàgina estàtica index3.html que és troba a la carpeta exam que penja de l'arrel de l'arbre de directoris del servidor web. S'ha d'indicar mètode, fitxer, protocol i versió del protocol.**
   GET /exam/index3.html HTTP/1.1
   Mètode: GET, fitxer: /exam/index3.html i protocol HTTP versió 1.1

4. **Indica l'esquema URI per accedir a un fitxer que s'anomena ex1m08uf1.pdf que es troba a la carpeta /home/daw2/examen d'un servidor FTP al qual s'ha d'accedir amb el login daw i password m08uf1pr1. El nom del servidor és ftp.fjeclot.edu i escolta pel port 21.**

   ftp://daw:m08uf1pr1@ftp.fjeclot.edu:21/home/daw2/examen/ex1m08uf1.pdf

5. **Indica quines són les diferències existents entre utilitzar un mètode POST i un mètode PUT**

   POST:
   - RFC 7231 Pàg. 25 4.3.3 → The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics. És a dir, el mètode POST demana que el recurso apuntat, que podria ser per exemple un programa en PHP, processi les dades incloses en la petició d'acord a la pròpia semàntica del recurs, o sigui, si tornem a l'exemple del programa en PHP, que processi les dades d'acord amb les instruccions incloses dins del programa.
   - El mètode POST es pot utilitzar per crear nous recursos dins del servidor però també pot servir per actualitzar-los. El mètode POST per tant canvia l'estat del servidor. La tendència majoritària és utilitzar-lo per crear nous recursos amb noves dades.
   - Funcions del mètode POST serien per exemple, enviar dades d'un formulari HTML a un programa pel seu processament, crear un nou recurs com per exemple un fitxer en el servidor, o actualitzar el fitxer.
   - El programa que ha de processar les dades s'indica a la línia d'estat (status) i les dades (o entity-body) es trobaran dins del body del missatge.
   - El codi de la resposta del servidor hauria de ser un 201 si s'ha creat un nou recurs, però també pot ser un 200 per dir al client que posi la resposta a la caché, o un 303 el resultat de processar la petició dóna com a resultat un recurs ja existent. Per diversos motius podria enviar-se un 206, 304 o 416.
   - El mètode POST no és idempotent. NO es garantitza que si repeteixo més d'una vegada la petició no es generin tants recursos com peticions s'han fet amb URIs. Per tant, el mètode POST no es repetible amb seguretat.

   PUT:
   - RFC 7231 Pàg. 26 4.3.4 → The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. És a dir, el recurs indicat a la línia de petició s'ha de crear o actualitzar amb les dades (entity-body) incloses al cos del missatge. En unes altres paraules, les dades s'han d'emmagatzemar a la localització demanada.
   - El mètode PUT es pot utilitzar per crear dins del servidor un nou recurs o actualitzar completament (reemplaçar) un recurs existent dins del servidor. El mètode PUT per tant canvia l'estat del servidor. La tendència majoritària és utilitzar-lo per actualitzar completament recursos amb noves dades.
   - D'acord amb el document RFC s'utilitza per fer una actualització completa (reemplaçar) de manera que s'ha d'anar amb compte amb com es fan les actualitzacions parcials. Hi ha un nou mètode anomenat PATCH que s'hauria d'utilitzar per actualitzacions parcials.
   - A la línia d'estat (status) d'un mètode PUT ens trobarem l'adreça del recurs que volem crear/ actualitzar i al cos del missatge les dades pel nou recurs.
   - Funcions del mètode PUT serien per exemple, actualitzar completament un fitxer html enviar dades d'un formulari HTML a un programa pel seu processament, crear un nou recurs com per exemple un fitxer en el servidor, o actualitzar el fitxer.
   - El codi de la resposta del servidor si ha creat un nou recurs serà un 201. Si s'actualitza un recurs existent, la resposta serà 200 o 204. Hi ha altres respostes possibles per errors: 400, 409, 415.
   - El mètode PUT és idempotent. Es garantitza que si repeteixo més d'una vegada una mateixa petició, l'estat del servidor no canviarà perquè recurs en el servidor no canviarà. El mètode PUT és repetible amb seguretat. Altres mètodes idempotents són per exemple GET HEAD o DELETE.

6. **Indica quines són les parts de les quals es composa una entitat i quins és el propòsit de cadascuna d'elles.**

   * Entitat (Entity) a RFC 2616 o Representació a RFC 7231 = capçaleres amb metadades sobre les dades que es troben dins del cos del missatge + dades dins del cos del missatge. No totes les capçaleres són part d'una entitat (veure RFC 2616 i RFC 7231).

   * Capçaleres d'entitat (entity-headers) =Proporciona informació sobre el contingut del cos del missatge o sobre la la resposta o peticion incloses en la línia de petició (per exemple, informació sobre el recurs demanat).
   Línia en blanc = Separació entre les capçaleres del missatge HTTP i el seu cos. No totes les capçaleres són d'entitat.

   * Cos de l'entitat (entity-body) = Són les dades que es troben dins del cos (body) del missatge. És una part opcional del missatge HTTP, de longitud variable i que és a on es troben les dades que s'envien del client al servidor (per exemple, dades d'un formulari) i també a on es troben les dades que envia com a resposta un servidor al client. Per exemple, quan enviem dades d'un formulari amb el mètode POST o PUT, aquestes dades es troben al cos de l'entitat. Quan un servidor envia el codi HTML d'una pàgina web com a resposta a una petició GET del client, el codi s'envia dins del cos del missatge.

   * Exemple dins d'un missatge de resposta:

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2020 14:00:09 GMT
Server: Apache/1.3.27
Content-type: text/html
Content-Length: 170
Last-Modified: Sun, 18 Oct 2020 12:05:30 GMT        Capçalere d'entitat
```

```
<html>
   <head>
      <title>Contingut de la resposat</title>
   </head>
   <body>
       <h1> Hola, món!!!!!</h1>
   </body>
</html>                                              Cos de l'entitat
```

7. **Indica quins són els 5 grups de missatges que pot enviar un servidor al client i quins són els propòsits de cadascun d'ells.**
   1xx --> Informació: La petició ha estat rebuda i el procés s'està duent a terme.
   2xx --> Èxit: La petició s'ha rebut correctament, s'ha pogut entendre i s'ha acceptat.
   3xx --> Redireccionament: L'User-Agent (o sigui, el client que en general és el navegador) ha de fer una sèrie d'accions addicionals per completar la petició. Per exemple, si la pàgina web demanada a canviat de servidor, el client rep un missatge 301 i una nova URI. El client enviarà automàticament al nou servidor.
   4xx --> Error del client: La petició no s'ha fet correctament, o el recurs no existeix, o no es té autorització per accedir-hi, etc...
   5xx --> Error del servidor: El servidor és conscient de que ha comès un error o és incapaç de dur a terme el mètode demanat. Per exemple, el servidor enviarà el el codi 501 si no té suport (o no reconeix) el mètode que el client ha demanat en la línia de petició, o el 505 si el servidor no dóna suport per una determinada versió del protocol HTTP que el client ha demanat en la línia de petició.

8. **Indica quins mètodes farien que el servidor respongués amb el missatge 201.**
   Qualsevol mètode que serveixi per crear nous recursos dins del servidor, o sigui, els mètodes PUT i POST

9. **Respon amb verdader o fals les següents afirmacions:**
   a) **El mètode POST és més segur perquè envia dades dins de la URL. Raona la resposta.**
   El mètode POST no és un mètode segur i el mètode GET sí que ho és. Però la pregunta fa referència a seguretat vist des del punt de vista de si podem veure les dades enviades o no a la barra d'adreces del navegador. Des d'aques punt de vist la respostà és FALS. El mètode POST és més segur (en el sentit indicat) que el GET perquè envia les dades dins del cos del missatge no dins de la línia de petició i per tant no es poden veure a la barra d'adreces.
   b) **El mètode GET reenvia la informació quan es recarrega una web i el navegador hauria d'avisar-nos d'aquest fet. Raona la resposta.**
   FALS. Això ho faria si tornés a enviar la informació quan es recarrega la pàgina, però això no passa perquè el mètode GET és idempotent, i no causa cap efecte permanent a enlloc excepte a la pantalla de l'usuari. Per aquest motiu, no cal que el navegador ens doni cap avís.
   c) **Les peticions de tipus POST no poden ser desades a les adreces d'interès (bookmark)**
   VERITAT. Les respostes a una petició POST no es desen a les adreces d'interés perquè d'acrod amb les especificacions (documents RFC) del protocol HTTP 1.0 i 1.1, l'aplicació no té cap manera de saber si el servidor retornarà la mateixa resposta a la mateixa petició en el futur.
   d) **Una petició de tipus POST envia les dades dins del cos del missatge**
   VERITAT. Així està especificat als documents RFC que expliquen el funcionament del protocol HTTP.
   e) **Les peticions de tipus GET admeten qualsevol tipus de dades (text, imatge, vídeo, etc..)**
   FALS. El mètode GET envia les dades dins de la línia de petició juntament amb l'URI. La línia de petició treballa amb caràcters USASCII d'acord amb els documents RFC que han de ser interpretats pel servidor i l'aplicació web. També, d'acord amb aquests documents existeixen restriccions en el format de l'URI que un format d'àudio, vídeo o imatge no poden assegurar que es compliran. També poden haver problemes amb el límit de bits que accepta un servidor com a part de l'URI de la línia de petició. De manera que el mètode GET no s'hauria d'utilitzar per enviar dades que no siguin ASCII.

10. **Quin és el significat de la següent resposta d'un servidor:**
    HTTP/1.1 302 Found
    Location: http://www.example.org/index.php
    El recurs ha canviat de localització. La nova URI és http://www.example.org/index.php. Si l'User-Agent ho pot fer, hauria de connectar-se a aquesta nova localització automàticament per obtenir el recurs demanat a la línia de petició.

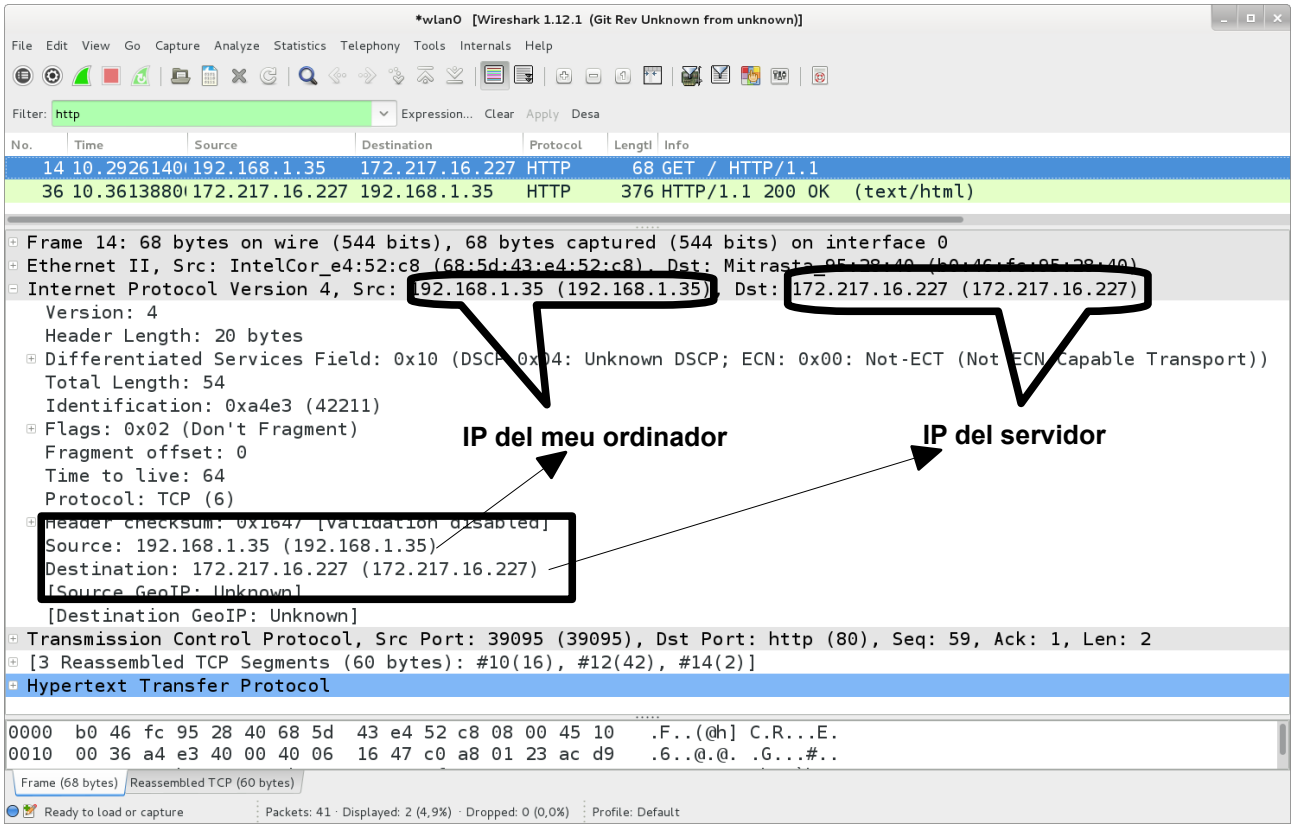
B) Peticions i respostes HTTP treballant amb Telnet i wireshark (50 %)

1. Posa en marxa **Wireshark** i filtra el contingut de les captures de manera que només es mostrin els continguts dels missatges HTTP. Fes la següent petició HTTP utilitzant: ***telnet www.google.cat 80***

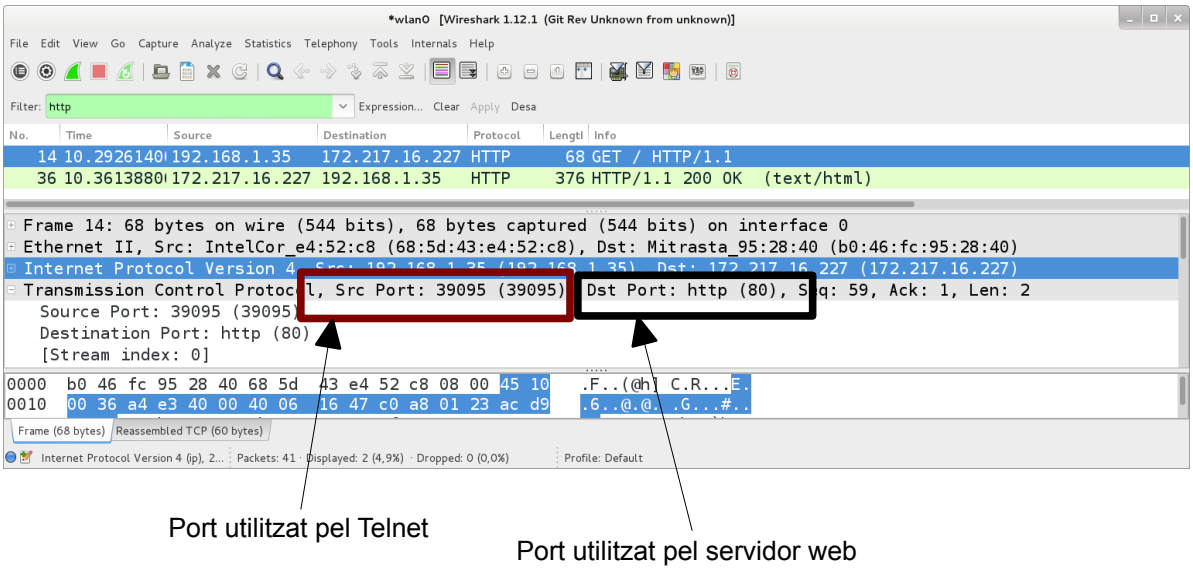
```
GET / HTTP/1.1
Host: www.google.cat
User-Agent: telnet
```

Mostra el resultat obtingut amb wireshark, i troba:

a) Dins de la capçalera IP, l'adreça IP del teu ordinador i la del servidor.

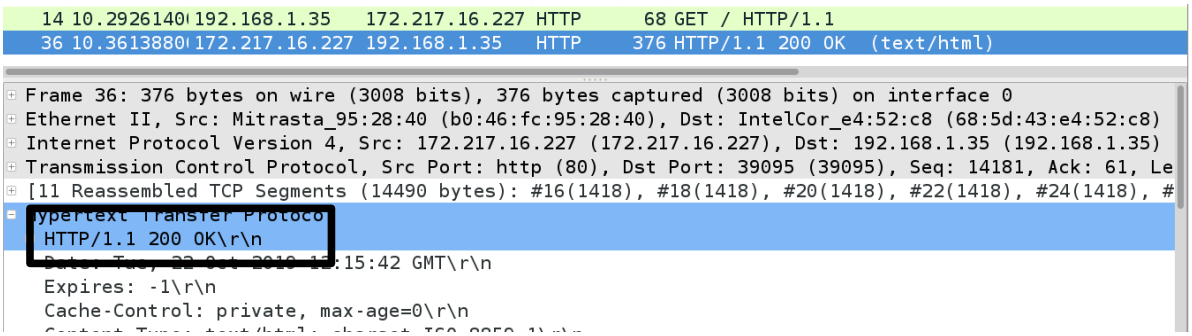


b) El port (tipus i valor) utilitzat pel programa telnet i pel servidor web



El programa Telnet utilitza el port 39095/tcp i el servidor web que utilitza el port 80/tcp

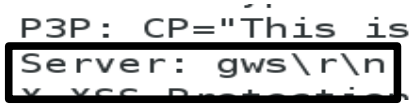
c) Quin codi de resposta dóna la capçalera HTTP enviada pel servidor? Què significa?



El codi de resposta és el 200 OK. Això vol dir, segons el document RFC7231 pàgina 51, que la petició s'ha realitzat amb èxit. El recurs demanat s'ha trobar i ha esta enviat.

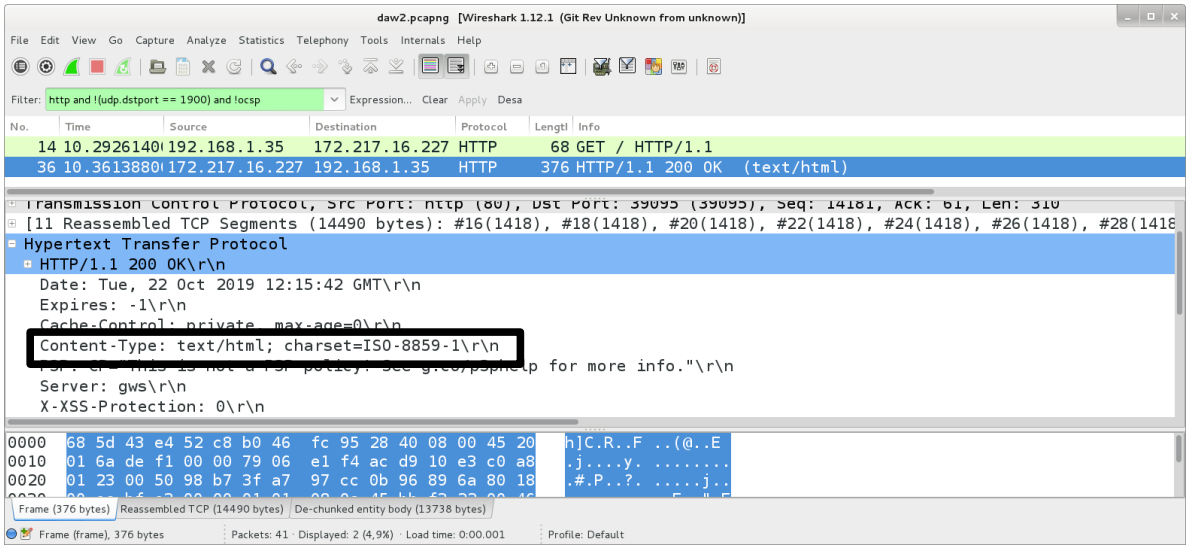
d) En el cas d'obtenir una capçalera *Location* a la resposta del servidor, explica el seu significat.
El servidor quan envia una resposta 302, també genera una capçalera de tipus "Location" amb l'URI de la nova localització, que hauria de fer servir l'User-Agent per fer un redireccionament automàtic. En aquest cas, l'User-Agent és el telnet, que no pot automàticament fer aquest redireccionament, tot i que ho podríem fer nosaltres mateixos manualment.

e) Que indica la capçalera *Server* de la resposta donada pel servidor? Explica el resultat que has obtingut.



El header de resposta Server d'acord amb el document RFC 7231 pàgina 72 té informació sobre el programari que utilitza el servidor per manegar la petició realitzada pel client. Aquesta informació pot ser utilitzada per l'User-Agent per millorar les comunicacions entre els dos programes com diu la documentació RFC. En aquest cas és el servidor gws (Google Web service), el servidor propi de Goole

f) Que indica la capçalera *Content-Type* de la resposta donada pel servidor? Explica el resultat que has obtingut.



El header de resposta Content-Type d'acord amb el document RFC 7231 pàgina 10 té informació sobre el tipus de dades que es troben dins del cos de l'entitat enviat pel servidor. En aquest cas, s'estan enviant text, que aquest text és un codi HTML, i que els caràcters estan codificats en versió ISO-8859-1, que permet codificar caràcters de la gran majoria dels sistemes d'escriptures del món (a diferència d'ASCII que té moltes limitacions). Amb ISO-8859-1 es poden codificar fins a 256 caràcters diferents necessaris per a l'escriptura de llengües d'Europa Occidental.

f) Comprova si la connexió es tanca immediatament després de rebre la resposta del servidor. Hi ha alguna manera de tancar la connexió?

La connexió no s'ha tancat. Encara podem enviar i rebre missatges HTTP entre el servidor i el client.

La connexió no es tanca immediatament. Podríem fer noves peticions o tancar la connexió enviant una petició errònia fent per exemple Ctrl-D o escrivint qualsevol cosa que no sigui protocol HTTP i que doni com a resposta un missatge 400 i tanqui la connexió També podem esperar a que es tanqui la connexió per part del servidor de manera automàtica després de passar un temps.

2. Fes la següent petició HTTP utilitzant: ***telnet www.google.com 80***

```
GET / HTTP/1.1
Host: www.google.es
User-Agent: telnet
Connection: close
```

Mostra el resultat obtingut amb wireshark, i respon a les següents preguntes:

a) Per què serveix la capçalera *Connection: close*? Què passa al executar la petició amb *Connection: close* a diferència de l'exercici anterior?

D'acord amb el document RFC 7230 apartat 6.1 pàgina 52, la capçalera Connection: close ja sigui dins d'una petició o d'una resposta, significa que el programa que envia aquest missatge tancarà la connexió un cop el procés de petició/resposta actual s'hagi completat. En aquest cas, això suposa que el client telnet es tancarà immediatament un cop rebuda la resposta del servidor i per tant no tindrem l'oportunitat d'intentar fer el redireccionament manualment.

```
HTTP/1.1 200 OK
Date: Tue, 30 Sep 2014 22:30:01 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=ID=0a784862c1cf7ef4:FF=0:TM=14121162
; path=/; domain=.google.es
Set-Cookie: NID=67=nl7OYgu0qh00Ti_hcjwYozRr3a84Ctl-yM
CeR0jC2ID00VoWW85SOg; expires=Wed, 01-Apr-2015 22:30:
P3P: CP="This is not a P3P policy! See http://www.goo
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic,p=0.01
Connection: close

<!doctype html><html itemscope="" itemtype="http://sc
 a la informaci□n mundial en castellano, catal□n, gal
bots"><meta content="/images/google_favicon_128.png"


"0bR_D-PH6ziPPbLDsk1_qVCMcTk"},"pcc":{}};goog
google.med('history');}});if(google.j&&google
>Connection closed by foreign host.
daniel@sh2:~$
```

Ara la connexió s'ha tancat immediatament un cop s'ha rebut la resposta del servidor

Una altre efecte és que en el cas anterior, el servidor enviava un missatge de resposta sense cap capçalera Connection, i en canvi, ara sí que envia una capçalera Connection amb l'opció close com es veu a la següent imatge:

```
HTTP/1.1 200 OK\r\n
Date: Tue, 30 Sep 2014 22:29:29 GMT\r\n
Expires: -1\r\n
Cache-Control: private, max-age=0\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
Set-Cookie: PREF=ID=e6fc45c663189aca:FF=0:TM=1412116169:LM=141211616
Set-Cookie: NID=67=P6Dq5brACd7HaD9DDNGanvUzKrCal80bGMLc3gslqvCKC0kCc
P3P: CP="This is not a P3P policy! See http://www.google.com/support
Server: gws\r\n
X-XSS-Protection: 1; mode=block\r\n
X-Frame-Options: SAMEORIGIN\r\n
Alternate-Protocol: 80:quic,p=0.01\r\n
Connection: close\r\n
\r\n
```

Per tant, el servidor també tanca la connexió un cop finalitza el procés de petició/resposta.

D'acord amb el document RFC7230, quan un client no té l'opció d'aprofitar una única connexió TCP per enviar més d'un missatge de petició i de rebre més d'un missatge de resposta del servidor, és a dir, no suporta connexions persistents, sempre hauria d'enviar un “close” per cada missatge de petició que envia. Igualment, un servidor que no té suport per treballar amb connexions persistents, hauria d'enviar un close cada vegada que envia un missatge de resposta (sempre i quan, el missatge no sigui del tipus 1xx)

b) Fixa't que a la petició el *Host* ara és www.google.es. Això ha fet que la resposta sigui diferent? Per què? Mostra la resposta i comenta el resultat obtingut i les diferències amb l'exercici anterior.

La capçalera de resposta per l'exercici anterior és aquesta (poden variar les dades, dates i hores):

```
HTTP/1.1 200 OK
Date: Wed, 23 Oct 2019 09:24:33 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2019-10-23-09; expires=Fri, 22-Nov-2019 09:24:33 GMT; path=/;
 domain=.google.cat; SameSite=none
Set-Cookie: NID=189=DBctRaxMoaP5vdJdUrWd3NjP8nvn1BAgp2V-CATwO3-LXYl_8-UMAnkHd2UC
FzwhBPiscKE8hrawjh_KnqzL6pZJWuK2CTuyE8djSHd44Epi0TxhnkvwlwBvXfSi027hk7Tc3jdTE5GQ
MiTNXuVktxlWBbRfrDr1vo7Oqcki3mY; expires=Thu, 23-Apr-2020 09:24:33 GMT; path=/;
domain=.google.cat; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

i per aquest exercici (poden variar les dades, dates i hores) és :

```
HTTP/1.1 200 OK
Date: Wed, 23 Oct 2019 09:25:47 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2019-10-23-09; expires=Fri, 22-Nov-2019 09:25:47 GMT; path=/;
 domain=.google.es; SameSite=none
Set-Cookie: NID=189=QdKrjSYj1hmLZ-c0y5D4yR6mJDCH01dYLl7le3nBvU0ghjRvj9B1hihwc0YU
GVq-Au8SRdhc_3JKlBjng_dSGu11ghaNY7JhNIlcLgZf5T9mUaG8jC6PcHIL9ULRrnKFgqMPA2mPs2EG
Q8AlxaMrIHMjl9FrwtbTIbaR3TQX-tw; expires=Thu, 23-Apr-2020 09:25:47 GMT; path=/;
domain=.google.es; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Connection: close
```

Observem que hi ha 2 canvis:

- S'ha creat una nova cookie amb valors diferents com per exemple el domini o el seu identificador
- El servidor ha afegit el camp de capçalera Connection amb el valor close de manera que es tanca la connexió després de rebre el cos del missatge de resposta.

També observem que encara que la connexió telnet sigui a www.gogole.com el servidor ens dóna la pàgina principal de www.google.es. Que això passi no ens hauria és normal perquè:

- Quan s'executa l'ordre telnet www.google.com 80, el valor www.google.com via DNS es converteix a una IP d'un servidor amb el qual ens de connecar. Realment l'ordre telnet no demana un domini en concret, simplement estableix una connexió amb un servidor via IP.

- Dins de la petició utilitzem la capçalera Host i aquí sí que indiquen l'ordinador i domini amb el que volem treballar. La capçalera Host és molt important si volem que un únic servidor web pugui donar resposta a més d'un nom d'un domini (google.cat, google.com, google.es, google.co.uk,etc....). Si el client envia un camp de capçalera Host i en el servidor activem un mecanisme del programa servidor (Apache2, IIS.....) anomenat Virtual Host, llavors podem fer que un únic servidor pugui sevir per donar respostes a peticions realitzades a més d'un nom domini i això permet que un únic servidor pugui ser per hostatjar múltiples dominis.

La capçalera Host és molt important i un client ha d'enviar-la sempre dins de tots els missatges de petició de recurs HTTP 1.1.

El valor Host també permet enviar un número de port. Podríem posar per exemple:

Host: www.google.es:80

Per més informació sobre el camp de capçalera Host, podeu llegir el document RFC 7230, apartat 5.4, pàgines 44 i 45.

c) Què significat pel navegador que el Expires valgui -1 i que el camp max-age de Cache-Control sigui igual a 0?

Expires es troba definit al document RFC 7234, secció 5.3 i pàgina 28. Cache-Control es troba definit al document RFC 7234, secció 5.2 i pàgina 2. El camp max-age està definit a la pàgina 26 dins de la secció 5.2.2.8 del document RFC 7234.

Quan un servidor envia un recurs al navegador (en aquest cas la pagina inicial de www.google.es), s'emmagatzema a la caché del navegador perquè la propera vegada que es demani no calgui connectar-se amb el servidor i es pugui agafar des de la pròpia caché del

navegador i presentar-lo a l'usuari. El problema és que si la web es canvia al servidor i el navegador no ho sap, aquest últim continuarà mostrant a l'usuari una pàgina que ja està obsoleta. Per evitar aquest problema, quan el servidor envia un recurs també envia una data d'expiració de la resposta (i per tant, del recurs). En el moment que passi la data de l'expiració, el navegador ja no podrà mostrar directament la pàgina emmagatzemada a la seva caché, i l'haurà de validar amb el servidor abans de poder-la mostrar a l'usuari.

Si el servidor envia una capçalera Expires amb un valor que no és una data, com per exemple -1, llavors s'obliga al navegador a considerar que el recurs ja està obsolet, encara que tot just el navegador l'acabi de rebre. Així doncs, sempre que l'usuari demani la pàgina inicial de google.es, el navegador haurà de validar la pàgina emmagatzemada a la seva caché amb el servidor abans de presentar-la. Com que la pàgina inicial de www.google.es sempre s'ha d'anar a validar amb el servidor llavors, que sempre que hi hagi canvis, l'usuari els podrà veure i la resposta donada mai estarà desactualitzada.

El camp max-age és un valor en segons que indica el temps que és pot considerar que el recurs enviat pel servidor no ha expirat i per tant encara és vàlid i es pot recuperar de la caché. En el cas de que max-age sigui igual a 0, vol dir que el recurs sempre està obsolet. Així doncs, sempre que l'usuari demani la pàgina inicial de google.es, el navegador haurà de validar la pàgina emmagatzemada a la seva caché amb el servidor abans de presentar-la.
Si la directiva max-age del camp Cache-Control i el camp de capçalera Expires es troben dins mateix missatge de resposta (com en el cas de la pràctica), llavors la directiva max-age invalida el camp de capçalera Expires.

NOTA: S'ha d'anar en compte de no confondre max-age d'un missatge de resposta enviada pel servidor amb el max-age enviat dins d'una petició per un client (RFC 7234, secció 5.2.1.1, pàgina 22). Quan ho envia un client, està indicant que no vol acceptar una resposta a una petició que tingui una edat més gran que la especificada en max-age i que tampoc vol acceptar un resposta caducada.

3. Posa en marxa **Wireshark** i amb l'eina **telnet** connectat al port **80/tcp** del servidor **xtec.gencat.cat** i realitza la següent petició:

```
HEAD  /ca/curriculum/professionals/  HTTP/1.1
Host: xtec.gencat.cat
User-Agent: telnet
Connection: close
```

Amb wireshark comprova:

a) Quina reposta dóna el servidor. Indica el significat del codi de la reposta i del camp Location.

```
HTTP/1.1 301 Moved Permanently
Server: AkamaiGHost
Content-Length: 0
Location: https://xtec.gencat.cat/ca/curriculum/professionals/
Cache-Control: max-age=0
Expires: Fri, 28 Oct 2022 10:44:35 GMT
Date: Fri, 28 Oct 2022 10:44:35 GMT
Connection: close

Connection closed by foreign host.
```

El codi de resposta és el 301 Moved permanently i per tant d'acord amb el document RFC 7231 apartat 6.4.2 el recurs ha canviat d'URI i el client (el navegador) si té aquesta capacitat, hauria de fer un re-link automàtic a la nova adreça que es troba al camp Location.

b) Indica quina és la nova adreça del servidor.
De fet el servidor no ha canviat d'adreça però ara utilitza https i la connexió pel port 80/tcp es redireccionada cap el port 443/tcp per treballar amb https.

c) Indica el programa servidor de pàgines web utilitzat per xtec.gencat.cat.
AkamaiGhost

d) Quina és la diferència entre HEAD i GET?
De fet no hi ha cap diferència important (excepte el Server-Timing) perquè el programa telnet no té capacitat per fer un re-link automàtica a la nova adreça URL.

4. Analitza la següent "conversa" HTTP entre un client i un servidor i descriu allò que està passant:

```
GET /private/index2.html HTTP/1.1
Host: localhost
```

El client demana amb el mètode GET el recurs index2.html que es troba al directori private que penja de l'arrel de l'arbre de directoris que utilitza el servidor per emmagatzemar els recursos. La comunicació amb el servidor utilitzarà la versió 1.1 del protocol HTTP. El valor Host és localhost, de manera que s'està intentant realitzar una connexió a un servidor que s'executa en el mateix ordinador des del qual s'executa el client.

```
HTTP/1.1 401 Authorization Required
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:18:15 GMT
WWW-Authenticate: Basic realm="Secure Area"
Content-Type: text/html
Content-Length: 311
```

El servidor envia el codi 401 (veure RFC 7235, secció 3.1, pàgina 6) en la línia d'estat del missatge de resposta. Això significa que la petició no s'ha dut a terme perquè manquen les credencials d'autenticació necessàries per accedir al recurs demanat. El servidor ha d'enviar obligatòriament el camp de capçalera WWW-Authenticate (RFC 7235, secció 4.1, pàgina 7) indicant un o més mètodes d''autenticació (en aquest cas només el mètode "Basic" o mètode d'autenticació d'accés bàsic) i els paràmetres (realm="Secure Area") necessaris per poder accedir al recurs demanat quan es torni intentar accedir novament des del client. El mètode "Basic" és una tècnica molt simple, sense encriptació i s''utilitza generalment sobre HTTPS. Quan s'utilitza un navegador com a client, en el moment de rebre la resposta apareixerà una finestra emergent (pop-up) per poder escriure les credencials (usuari i contrasenya). El realm defineix un conjunt de recursos del servidor que estan protegits i requereixen credencials per accedir-hi. Tots els recursos dins d'un realm comparteixen les mateixes credencials.

```
GET /private/index2.html HTTP/1.1
Host: localhost
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

L'usuari escriu les credencials dins de la finestra emergent i el client genera una nova petició amb les dades donades pel client, indicant el mètode d'autenticació i a continuació les credencials codificades (que no vol dir encriptades) en **Base64** en comptes d'**ASCII** dins del camp Authorization (RFC 7235, secció 5.3.5, pàgina 8)

```
HTTP/1.1 200 OK
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:19:07 GMT
Content-Type: text/html
Content-Length: 10476
```

Les credencials enviades són correctes i el servidor envia el codi 200 OK (RFC 7231, secció 6.3.1, pàgina 51) que la petició del client s'ha rebut, entès i acceptada, i que per tant s'envia el recurs dins del cos del missatge de resposta. El programa servidor és HTTPd versió 1.0. El missatge de resposta s'ha creat el 27 de Novembre del 2004 a les 10:19:07 GMT. El contingut és de tipus text/html (veure RFC 2854), o sigui text amb el conjunt de caràcters UTF-8 preferiblement i el contingut és HTML, i la mida del cos del missatge és de 10476 bytes. Altres tipus de contingut podrien ser per exemple imatges, àudio, vídeo, PDF, gzip. Per veure una llista de tipus de continguts pots anar a: http://en.wikipedia.org/wiki/Internet_media_type#Type_text .

C) Peticions i respostes HTTP treballant amb Firefox i wireshark (20 %)

NOTA: Abans de començar aquesta part:

- Accedeix a Preferències → Privadesa i seguretat
- Neteja Galetes i contingut de memòria cau
- Neteja historial i configura Firefox per no recordar mai l'historial
- A la barra de localització (a on escrius les adreces URL) escriu about:config per accedir a la configuració.
- Busca l'entrada de configuració browser.urlbar.autoFill. Fes que el seu valor sigui False.
- **Reinicia Firefox**

1.- Treballant amb la capçalera *Accept-Language*

a) Des de "about:config", selecciona el paràmetre de configuració intl.accept_languages i indica que s'utilitzaran com idiomes de treball les següent llengües en aquest ordre: Català (ca), Spanish(es), English (en). Esborra l'historial complet del teu navegador. Tanca i torna obrir el teu navegador. Connecta't a http://www.collados.org. Comprova ara la capçalera de petició realitzada pel navegador. Indica el valor del camp **Accept-Language** enviat pel navegador dins de la capçalera.

```
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Host: www.collados.org\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: ca,en;q=0.7,es;q=0.3\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  \r\n
```

b) Quin és l'efecte del camp Accept-Language
D'acord amb el document RFC 7231, secció 5.3.5 i pàgina 42, aquest camp de la capçalera HTTP pot ser utilitzat per un User-Agent per indicar al servidor la (o les llengües) que són les preferides en la resposta del servidor.
Si es defineix una única llengua, llavors aquesta serà l'escollida com a llengua preferida. Si dins d'Accept-Language hi ha una llista de llengües, a cadascuna se li pot assignar un valor que representa la preferència de les llengües dins de la llista.
Una petició sense Accept-Language significa que el client accepta qualsevol llengua com a resposta i que totes tenen la mateixa preferència.
c) Quin és el propòsit del valor **q**?.
És un valor entre 0 i 1 (no ha de tenir més de 3 decimals). Com més gran és el valor associat, mes preferent és una llengua. Si no s'assigna cap valor, llavors la preferència està marcada per l'ordre de la llista.

2- Comprovant el funcionament de les peticions POST i GET

* Quan s'envien dades introduïdes dins d'un formulari amb el mètode POST, aquestes dades aniran dins del cos del missatge, en canvi quan s'utilitza un mètode GET, aquestes dades s'incorporen a la línia de petició del missatge.
* La resposta a un mètode GET és pot afegir a la memòria cau (o caché) del navegador. En determinades condicions molt específiques d'acord amb les indicacions de l'apartat 4.3.3 del document RFC 7231 i l'apartat 4.2.1 del document RFC 7234, la resposta del mètode POST no s'afegeix a la memòria cau. Molts navegadors no accepten afegir la resposta a un mètode POST a la memòria cau.
* La resposta a un mètode GET pot ser afegida als favorits i romandrà a l'historial del navegador. La resposta a un mètode POST no pot ser afegida als favorits i romandrà a l'historial del navegador.
* Les peticions de tipus POST no tenen restriccions de longitud i de tipus (també es poden enviar dades binàries). El mètode GET requereix que la petició sigui de tipus text (ASCII) i tindrà una restricció de longitud associada a la longitud màxima de la línia de resposta acceptada pel navegador o el servidor (normalment, per seguretat, no hauria de ser de més de 2048 bytes).
* El mètode POST no és idempotent a diferència del mètode GET.
* Quan s'utilitza el mètode POST i es refresca un formulari, les dades s'envien novament al servidor i el navegador haurà de mostrar un missatge a l'usuari informant d'aquest fet de manera que pugui acceptar-ho o no.
* Les dades d'un formulari enviades amb el mètode GET es poden visualitzar a la barra d'adreces del navegador i formen part de l'URL. Aquestes dades no es visualitzen a la barra d'adreces amb el mètode POST perquè es troben dins del cos del missatge.
* El mètode GET s'utilitza per demanar un recurs a un servidor a partir de la seva URI i no ha deproduir alteracions de les dades en el servidor. Si volem enviar dades a un servidor que pugin modificar l'estat del servidor (actualització/creació), el mètode a utilitzar hauria de ser el POST.

a) Treballa amb Firefox. Connectat a Connectat a **http://www.collados.org/daw2/m08/uf1/php_html_post_get/form.html**. Comprova el valor del camp User-Agent del primer missatge HTTP enviat des del client.

```
Hypertext Transfer Protocol
  GET /daw2/m08/uf1/php_html_post_get/form.html HTTP/1.1\r\n
    Host: www.collados.org\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: ca,en;q=0.7,es;q=0.3\r\n
    Accept-Encoding: gzip, deflate\r\n
    DNT: 1\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
```

D'acord amb la captura realitzada, el meu User-Agent és el Firefox 70.0 executant-se sobre una màquina que treballa amb una versió de Linux de 64 bits. Per més informació sobre el significat de la resta del camps, aneu a http://www.useragentstring.com/.

b) Accedeix a la web Mètode Post. Omple el formulari i tramet la consulta. Comprova:

- Amb wireshark que s'ha creat una petició de tipus POST. Demostra-ho.

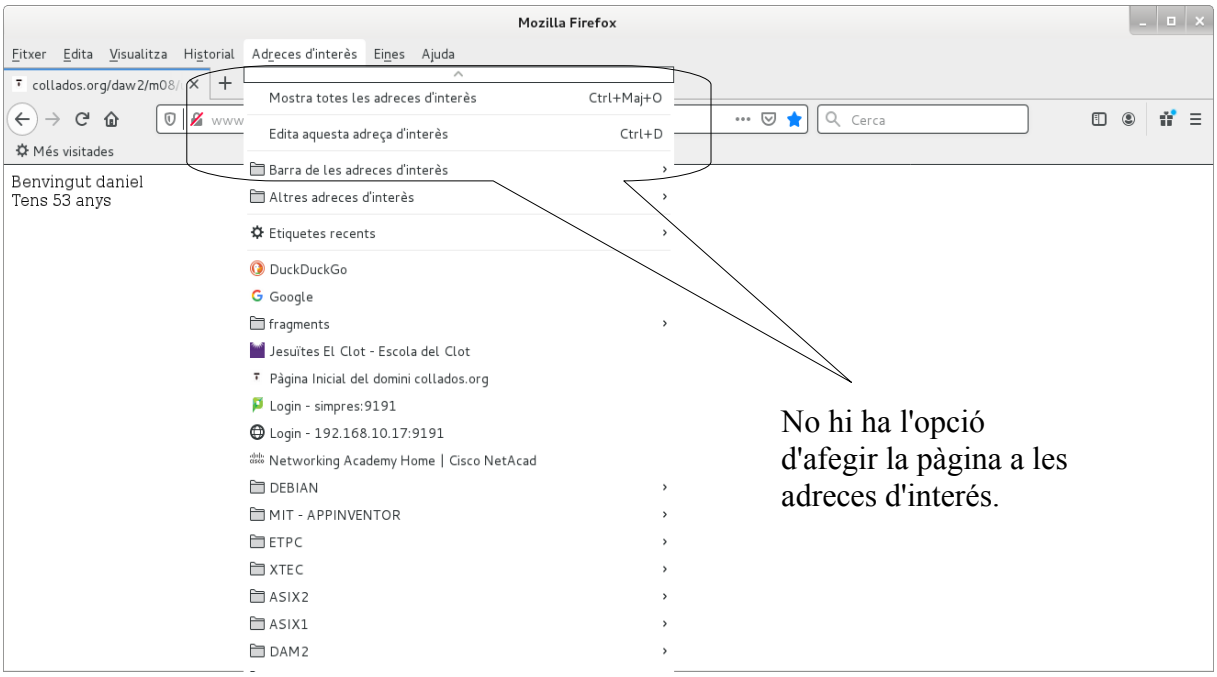
```
728 248.688320000 192.168.1.37      139.162.131.226   HTTP    630 POST /daw2/m08/uf1/php_html_post_get/post.php HTTP/1.1  (appli
732 248.807698000 139.162.131.226   192.168.1.37      HTTP     71 HTTP/1.1 200 OK  (text/html)

Frame 728: 630 bytes on wire (5040 bits), 630 bytes captured (5040 bits) on interface 0
Ethernet II, Src: CeLink_08:d0:0c (a0:ce:c8:08:d0:0c), Dst: Mitrasta_95:28:40 (b0:46:fc:95:28:40)
Internet Protocol Version 4, Src: 192.168.1.37 (192.168.1.37), Dst: 139.162.131.226 (139.162.131.226)
Transmission Control Protocol, Src Port: directplaysrvr (47624), Dst Port: http (80), Seq: 1, Ack: 1, Len: 564
Hypertext Transfer Protocol
  POST /daw2/m08/uf1/php_html_post_get/post.php HTTP/1.1\r\n
    Host: www.collados.org\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: ca,en;q=0.7,es;q=0.3\r\n
    Accept-Encoding: gzip, deflate\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  Content-Length: 18\r\n
```

- Comprova que les dades s'envien dins del cos del missatge. Demostra-ho.

```
Referer: http://www.collados.org/daw2/m08/uf1/php_html_post_get/post.php\r\n
Upgrade-Insecure-Requests: 1\r\n
\r\n
[Full request URI: http://www.collados.org/daw2/m08/uf1/php_html_post_get/post.php]
[HTTP request 1/1]
[Response in frame: 732]
HTML Form URL Encoded: application/x-www-form-urlencoded
Form item: "nom" = "daniel"
Form item: "edat" = "53"
```

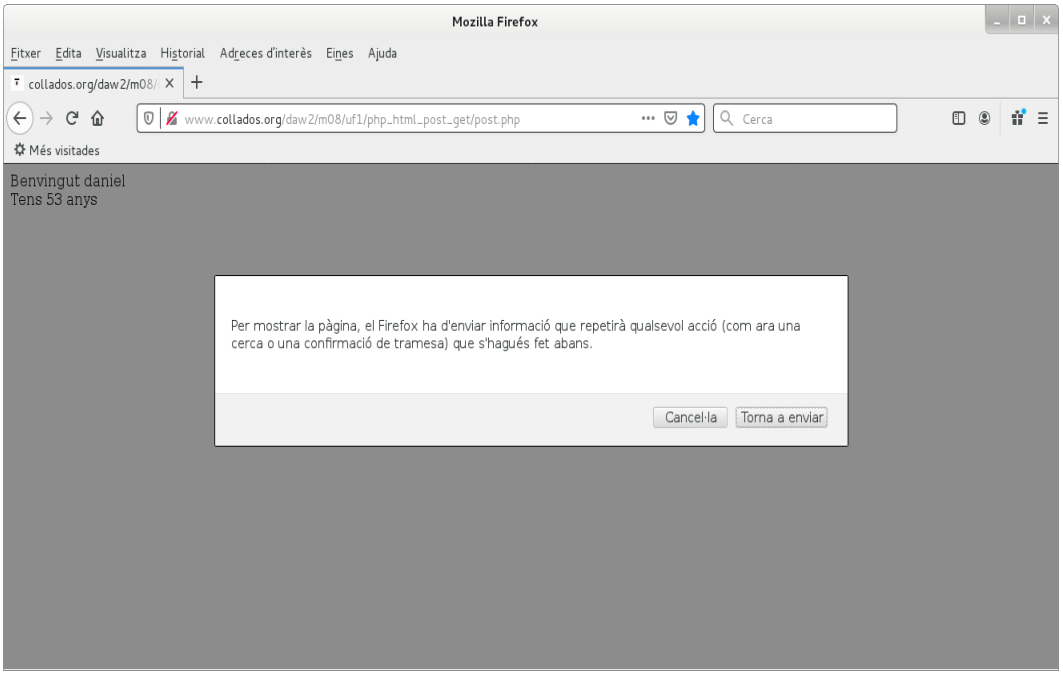
- Que les dades no són visibles a la barra d'adreces del navegador.

- Demostra que el resultat no es pot afegir a les adreces d'interès



- Comprova des de wireshark que quan recarreguem la pàgina es torna a enviar la petició i que el navegador dóna un missatge d'avís.

Des del navegador podem observar el següent comportament:



El navegador ens mostra un missatge d'avís i ens demana si acceptem tornar a enviar la informació.

D'altra banda, la captura amb wireshark mostra això en el moment de recarregar la pàgina:

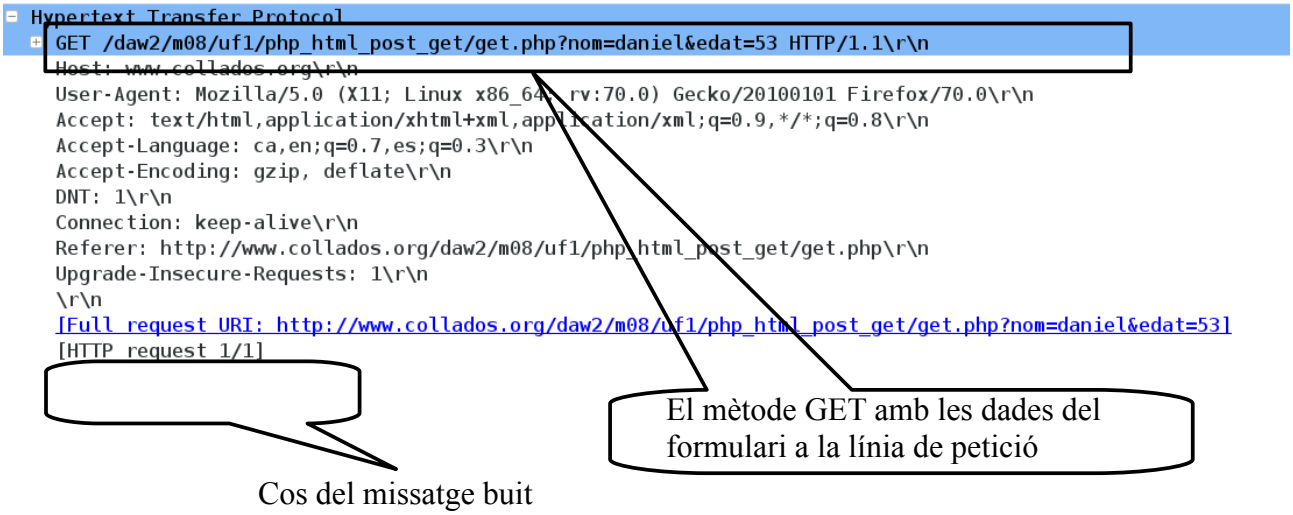
```
 728 248.688320000 192.168.1.37    139.162.131.226 HTTP  630 POST /daw2/m08/uf1/php_html_post_get/post.php HTTP/1.1  (appl
 732 248.807698000 139.162.131.226 192.168.1.37    HTTP   71 HTTP/1.1 200 OK  (text/html)
1168 886.832478000 192.168.1.37    139.162.131.226 HTTP  656 POST /daw2/m08/uf1/php_html_post_get/post.php HTTP/1.1  (appl
1172 886.958966000 139.162.131.226 192.168.1.37    HTTP   71 HTTP/1.1 200 OK  (text/html)
1317 1135.88957300( 192.168.1.37    139.162.131.226 HTTP  656 POST /daw2/m08/uf1/php_html_post_get/post.php HTTP/1.1  (appl
1321 1136.01249400( 139.162.131.226 192.168.1.37    HTTP   71 HTTP/1.1 200 OK  (text/html)
```

Observem que es torna a enviar la petició amb les dades al cos del missatge igual que si fos la primera vegada que s'ha fet la petició. En aquest cas estem 2 recarregues perquè he fet la petició 2 vegades, i veiem que igualment, a la darrera petició, es tornen a enviar les dades.

```
1317 1135.88957300( 192.168.1.37    139.162.131.226 HTTP  656 POST /daw2/m08/uf1/php_html_post_get/post.php HTTP/1.1  (appl
1321 1136.01249400( 139.162.131.226 192.168.1.37    HTTP   71 HTTP/1.1 200 OK  (text/html)
  Host: www.collados.org\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: ca,en;q=0.7,es;q=0.3\r\n
  Accept-Encoding: gzip, deflate\r\n
  Referer: http://www.collados.org/daw2/m08/uf1/php_html_post_get/post.php\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
⊞ Content-Length: 18\r\n
  Origin: http://www.collados.org\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Cache-Control: max-age=0\r\n
  \r\n
  [Full request URI: http://www.collados.org/daw2/m08/uf1/php_html_post_get/post.php]
  [HTTP request 1/1]
  [Response in frame: 1321]
HTML Form URL Encoded: application/x-www-form-urlencoded
⊞ Form item: "nom" = "daniel"
⊞ Form item: "edat" = "53"
```

c) Accedeix a la web Mètode Get. Omple el formulari i tramet la consulta. Comprova:

- Amb wireshark que s'ha creat una petició de tipus GET. Demostra-ho.

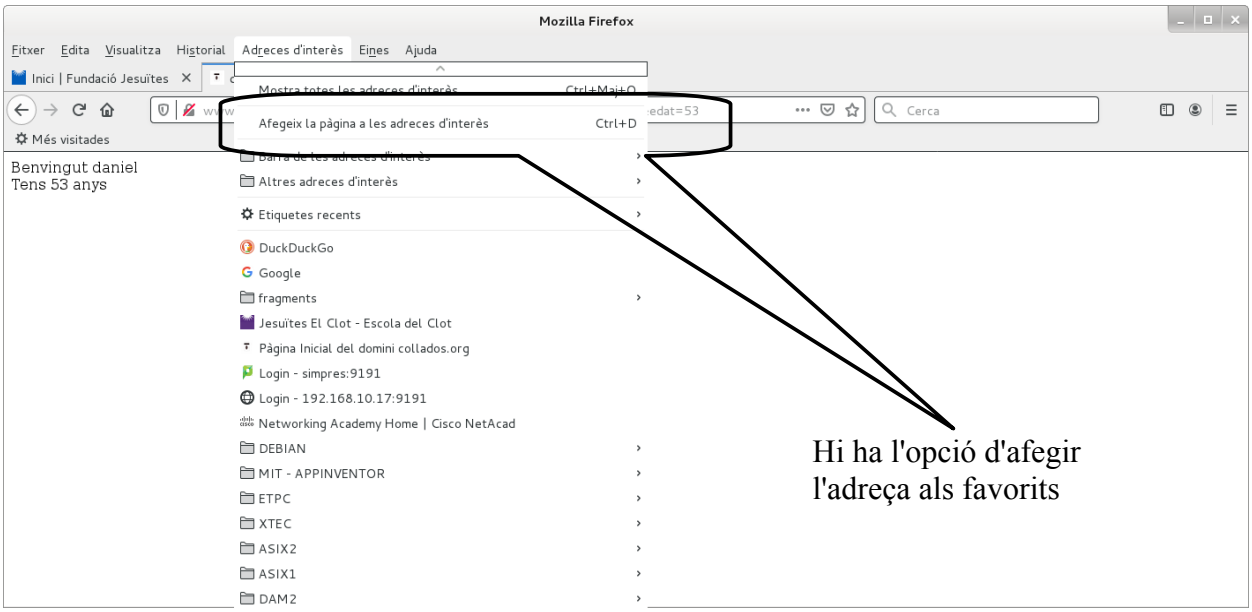


- Comprova que les dades s'envien dins de la capçalera del missatge. Indica dins de quin camp es troben aquestes dades. Demostra-ho.

  Com es pot veure a l'imatge anterior, les dades es troben a la línia de petició i formen part de l'URI.

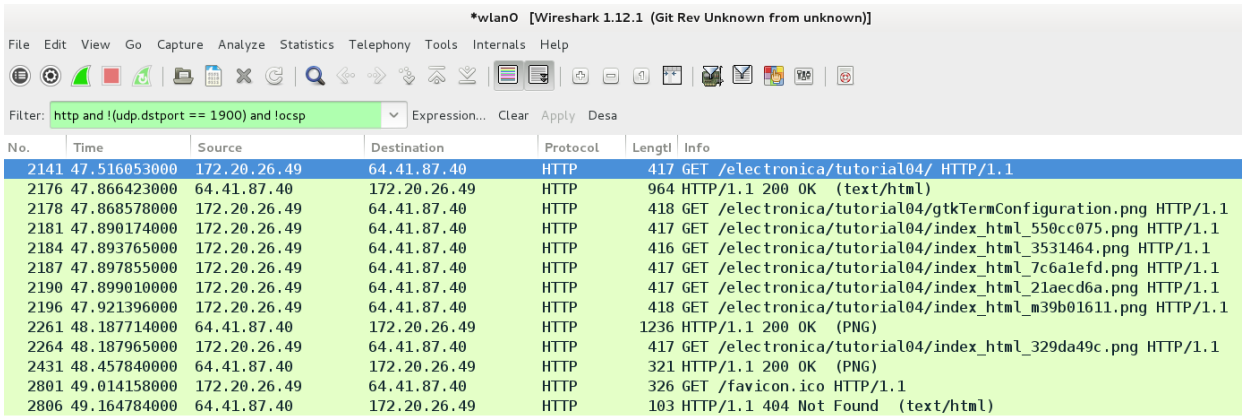
- Que les dades són visibles a la barra d'adreces del navegador.

- Demostra que el resultat es pot afegir a les adreces d'interès



3- Múltiples connexions

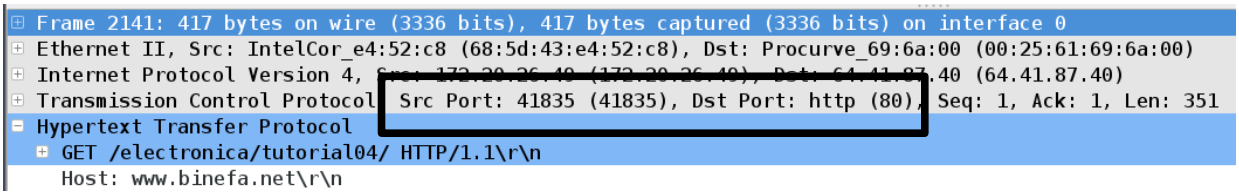
a) Treballa amb Firefox. Connectat a **http://www.binefa.net/electronica/tutorial04/**. Comprova quantes peticions GET s'han generat per part del client i i quantes respostes respostes ha enviat el servidor.

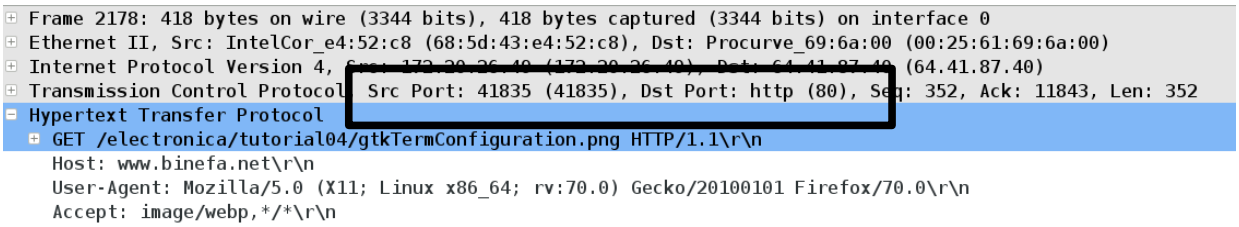


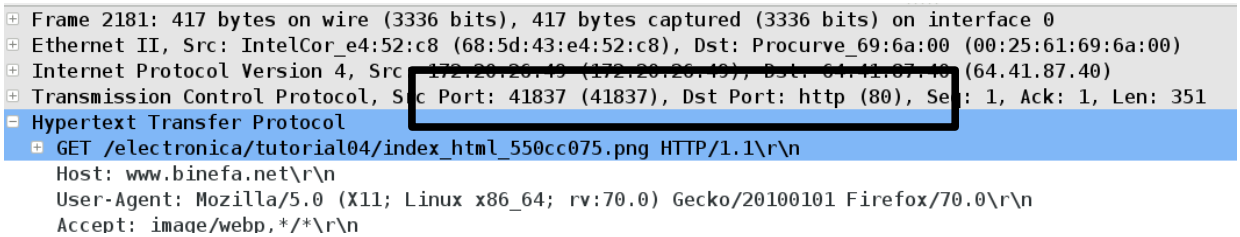
b) Indica el motiu pel qual, a part de la petició inicial del client i la resposta inicial del servidor, s'han generat noves peticions i respostes.

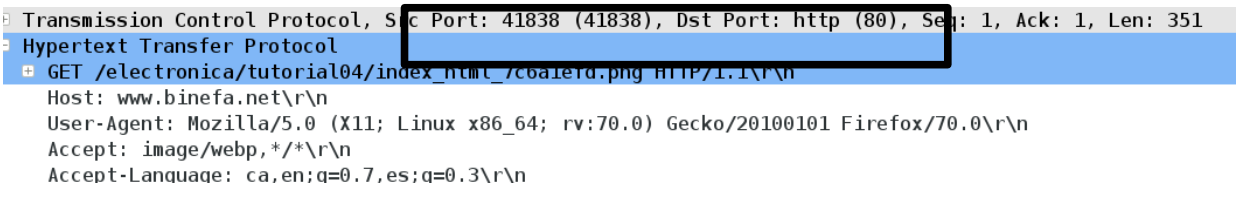
Si comprovem el codi HTML enviat al cos del missatge de la primera resposta, veuerem que es demanen una sèrie d'imatges en format **png**. El client demana al servidor aquestes imatges per mitjà de noves peticions utilitzant el mètodes GET perquè no arriben juntament amb el codi HTML i per això es generen aquestes peticions noves. El servidor envia també respostes amb les imatges en el cos del missatge, i per això es generen missatges de resposta nous.

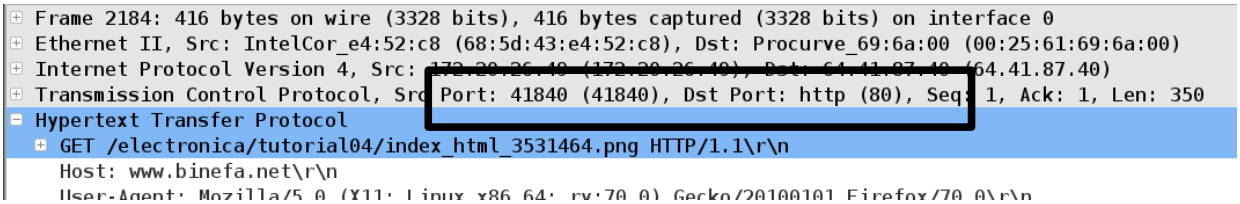
c) Comprova que cada petició ha implicat la utilització d'un nou port per part del client. Per què?











NOTA: No he fet captures de tots els GET enviat però poso aquests com a mostra de com va canviant el port per cada petició enviada al servidor per obtenir una imatge.

Estem treballant amb el protocol HTTP 1.1 de manera que no caldria obrir un port TCP per cada URI a la qual ens volguessin connectar. El protocol HTTP 1.1 (a diferència del 1.0) treballa amb connexions persistents, de manera que es pot aprofitar una única connexió TCP per recollir recursos de múltiples URIs. De fet, la primera petició, i la primera imatge fan servir el mateix port TCP com es pot comprovar a les captures realitzades. Treballar sense connexions persistents té el problema de que la xarxa es pot congestionar més fàcilment, augmenta el temps de resposta i la carrega de la CPU. Ara bé, si per cada imatge establim una connexió diferenet fent servir ports TCP diferents, podrem baixar més ràpidament les imatges del servidor, perquè podrem descarregar-les en paral·lel..

D) Taula comparativa de mètodes

a) Què significa que un mètode sigui segur?. Quins mètodes són segurs?
D'acord amb el document RFC 7231 apartat 4.2.1 els mètodes segurs son aquells que per la seva definició són esencialment de només lectura i que per tant, per ells mateixos, no esperem que facin cap mena de mdificació en l'estat del servidor i els seus recursos. El mètodes GET, HEAD, OPTIONS i TRACE són mètodes segurs.
b) Què significa que un mètode sigui "Cacheable"?. Quins mètodes són "Cacheables"?
D'acord amb el document RFC 7231 apartat 4.2.3 els mètodes "cacheables" són aquells que es permet que les seves respostes s'emmagatzemin per ser utilitzades en el futur. Un lloc típic a on emmagatzemar una resposta és a la caché d'un proxy o un navegador. En general els mètodes cacheables són GET, HEAD i POST però a la realitat, una majoria aclaparadora d'implementacions pràctiques de sistemes de cache només emmagatzemen respostes de tipus GET i HEAD.
c) Fes una taula comparativa dels mètodes GET, PUT, POST, HEAD i DELETE indicant per cada mètode si té les següents propietats:

- La petició envia dades al cos del missatge?
- La resposta a la petició té dades al cos del missatge?
- És un mètode segur?
- És un mètode idempotent?
- És un mètode cacheable?

Mètode HTTP	Petició amb dades al cos del missatge	Resposta amb dades al cos del missatge	Segur	Idempotent	Cacheable
GET	NO(*2)	SÍ	SÍ	SÍ	SÍ
POST	SÍ	SÍ	NO	NO	SÍ(*1)
PUT	SÍ	SÍ	NO	SÍ	NO
HEAD	NO(*2)	NO	SÍ	Sí	SÍ
DELETE	NO(*2)	SÍ	NO	Sí	NO

(*1) Però només en condicions molt específiques indicades a RFC 7231 4.3.3. A la pràctica, gairebé ningú emmagatzema a la caché les respostes a peticions del mètode POST.
(*2) Estrictament parlant, l'enviament és opcional si llegin els documents RFC però, a la pràctica, normalment NO s'envien dades al cos del missatge.

Per veure una taula completa amb tots els mètodes HTTP, aneu a:

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Safe_methods