Curs 6

HTTP

Ce se întâmplă când scriem în browser *www.cs.ubbcluj.ro* și se dă enter?

→ browser-ul (clientul) se conectează la serverul web *www.cs.ubbcluj.ro* pe portul 80 (prin HTTP) sau 443 (prin HTTPS)

Ca să nu țină resurse ocupate, de fiecare dată când un client se conectează și cere ceva, după deservirea clientului conexiunea se închide.

Ca un server web să deservească mai mulți clienți în paralel, se creează thread-uri pentru fiecare. Dacă vin foarte mulți clienți în paralel, nu are rost să țină procesele live și running mereu, deoarece poate utilizatorul nu face nimic o perioadă de vreme pe pagină.

telnet www.cs.ubbcluj.ro 80 → încercăm să comunicăm cu serverul, noi fiind clientul

!!! Faptul că are www în nume sugerează că e un server web.

GET / HTTP/1.1

HOST: www.cs.ubbcluj.ro

newline → semnalează faptul că nu mai avem alte headere de trimis

→ a doua linie face parte dintr-un antet HTTP pe care clientul (browserul) îl trimite pe request serverului

→ serverul web afișează înapoi codul sursă al paginii web

GET → metodă prin care se face o cerere

/ → numele resursei care se dorește a fi cerută prin metoda GET (ar fi / din capătul www.cs.ubbcluj.ro/)

HTTP/1.1 → versiunea de protocol

!!! Există și HTTP 2 care vine cu unele chestii mai noi: suport mai bun pentru web sockets, renunță puțin la caracterul stateless, păstrează conexiunile deschise mai bine, suportă mai bine partea de streaming etc.

Răspunsul vine sub forma HTTP/1.1 response_code (status_code) value_for_code

ex: HTTP/1.1 200 OK

HTTP/1.1 301 Moved Permanently

Coduri: 200 (ok), 3xx (redirectări – moved permanently (301)), 4xx (erori client-side – not found (404), forbidden (403)), 5xx (erori server-side – internal server error (500))

!!! Header-ele preced orice conținut pe care îl trimite serverul web înapoi clientului (pe response), apoi e un newline, iar după apare conținutul ce poate fi văzut.

Exemplu de header-e ce vin pe response:

```
HTTP/1.1 301 Moved Permanently
Date: Wed, 31 Mar 2021 08:10:16 GMT
Server: Apache
Location: http://www.cs.ubbcluj.ro/~bufny/
Content-Length: 240
Content-Type: text/html; charset=iso-8859-1
```

Content-Length: numărul de octeți al conținutului ce vine după newline:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.cs.ubbcluj.ro/~bufny/">here</a>.</p>
</body></html>
```

Content-Type ajută browserul să afișeze conținutul respectiv (e obligatoriu). La o imagine, valoarea acestui header va fi *image/jpeg* (de ex.)

De ce trebuie precizat HOST: www.cs.ubbcluj.ro (care zice că resursa menționată prin GET să fie adusă de pe serverul respectiv), pentru că oricum ne-am conectat la server prin telnet?

→ Pe un server web pot fi găzduite mai multe site-uri (domenii).

www.firma1.ro → 1.2.3.4

www.firma2.ro → 1.2.3.4

Vine un client și vrea să navigheze pe firma1.ro. Se face conexiune pe 1.2.3.4:80. Vine alt client și se conectează pe firma2.ro, prin 1.2.3.4:80. În acest moment avem doi clienți diferiți care, din punct de vedere conceptual, nu fac nimic diferit. Serverul web ar trebui să facă distincție între ce vor clienții care se conectează.

!!! În momentul conexiunii prin telnet, nu ne-am conectat la www.cs.ubbcluj.ro pe portul 80, ci ne-am conectat la adresa IP 193.0.225.34 pe portul 80.

Pentru că în situațiile alea două, până în momentul conectării, serverul nu își dă seama ce vrea clientul (vrea firma1.ro sau firma2.ro), clientul trebuie să ceară explicit prin header-ul HOST de la ce site vrea informații.

Dacă navigăm pe baza IP-ului, ar trebui să meargă sau nu?

→ Nu e obligatoriu să meargă. Dacă facem un request către un anumit nume de server web, merge, însă dacă facem prin adresa lui IP s-ar putea să nu meargă, deoarece pe serverul web cu IP-ul respectiv sunt găzduite domenii diferite.

!!! Header-ele nu sunt afișate în browser, le putem vedea prin developer tools.

!!! La nivelul serverului, fiecare client care se conectează și face câte un request este diferit. Trebuie asociate toate conexiunile care vin de la clienții diferiti aceleiași sesiuni de lucru. O modalitate era să ne uităm la adresa IP a clientului (stinky) – nu prea merge.

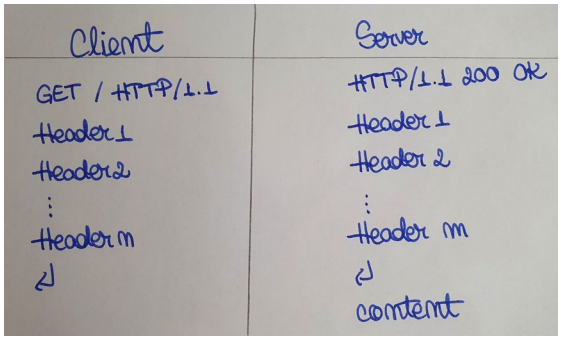
Ca să se asocieze o sesiune de lucru, e nevoie de un cookie de sesiune – e o bucățică de informație pe care browserul o tot trimite serverului și serverul o trimite înapoi browserului (prin header-ul Set-Cookie).

!!! O sesiune HTTP ține cât vrem noi la nivelul backend-ului.

Dacă furăm un cookie de sesiune și îl inserăm în browserul nostru ca să facă un request cu acel cookie, practic putem ajunge în aplicație unde se află utilizatorul al cărui cookie am furat.

Curs 7

La cererea prin GET, nu există nimic după header-ele de pe request. Response-ul prin GET: serverul îi dă linia inițială HTTP/1.1 ..., după vin headere-le pe response (ex: Content-Type, obligatoriu, Set-Cookie, Date etc.), după vine conținutul.



!!! Din perspectiva GET-ului, response-ul seamănă cu request-ul, cu excepția conținutului primit de la server.

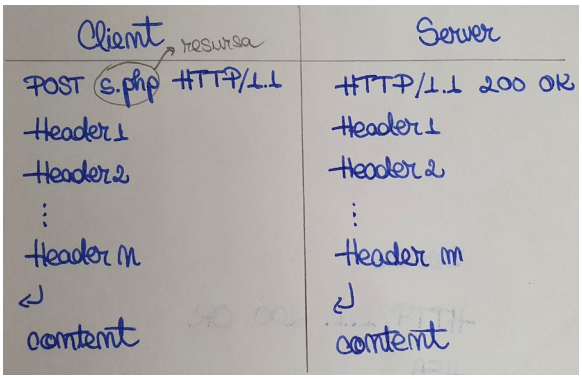
Prin GET zicem serverului să ne dea ceva. Datele pe care le trimite clientul la server se pun în query string și fac parte din numele resursei care e specificată în linia inițială de GET (GET resource HTTP/1.1)

!!! 90% din request-urile care le face browserul spre server sunt GET-uri.

Toată informația de la client nu poate fi luată de bună, header-ele pot fi modificate.

Dacă facem submit unui formular prin POST, perechile de parametrii vor fi încapsulate în *content* și se trimit ca și body-ul request-ului. Nu mai există query string.

Conținutul care se trimite pe request poate fi și un XML, o expresie JSON etc.



Dacă se face submit prin POST, se presupune că va avea loc o alterare a datelor pe backend. Această alterare poate avea loc și prin GET, deși GET ar trebui să fie *read-only*.

!!! O altă diferență între GET și POST, la resursele și URL-urile din POST nu putem face bookmark. La cele prin GET se poate, deoarece toată informația de care e nevoie ca să se genereze conținutul care se formează pentru response, se regăsește în numele resursei (query string-ul) de după.

Alte metode:

- HEAD – arată ca un GET, doar că pe request sunt doar headere, iar pe response sunt, la fel, doar headere (în cazul GET-ului aveam conținut și după newline-ul de pe response) – util dacă vrem să cerem ceva la server, dar nu ne interesează conținutul, ci header-ele

- PUT – seamănă cu un fel de POST (POST se folosește și pentru crearea de resurse, iar PUT pentru modificarea resurselor)
- DELETE – șterge o resursă de pe server (nu putem șterge resurse de pe backend prin HTTP – motiv de securitate); are doar headere (nu și conținut), iar pe response sunt, la fel, doar headere
- CONNECT – folosită în contextul folosirii protocolului HTTPS