

Corso di Laurea in Ingegneria Informatica



**Corso di Reti di Calcolatori
(a.a. 2011/12)**

Roberto Canonico (roberto.canonico@unina.it)

Giorgio Ventre (giorgio.ventre@unina.it)

Protocolli applicativi: HTTP

3 ottobre 2011

**I lucidi presentati al corso sono uno strumento didattico
che NON sostituisce i testi indicati nel programma del corso**

Nota di copyright per le slide COMICS



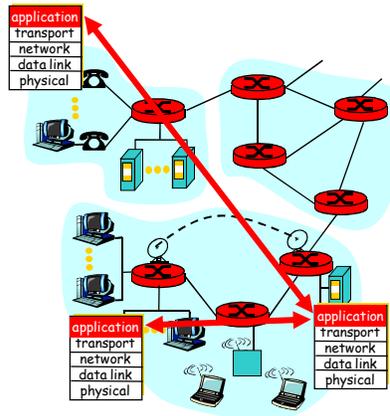
Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,
Marcello Esposito, Roberto Canonico, Giorgio Ventre

Applicazioni e protocolli dello strato applicativo



3

Strato dell'Applicazione



- Comunicazioni tra processi
- Cosa definisce un protocollo dello strato di applicazione:
 - tipo di messaggi scambiati
 - sintassi dei messaggi
 - semantica dei campi
 - regole di trasmissione
- Applicazioni lato client e lato server (Es.: *servizi Web, FTP*)

4

Comunicazione tra processi



- API: Application Programming Interface
 - definisce l'interfaccia tra il livello applicativo e il livello trasporto
 - socket: Internet API
 - due processi comunicano mandando dati in un socket, e leggendo dati dal socket
- Come un processo può "identificare" l'altro processo con cui vuole comunicare?
 - indirizzo IP dell'host che esegue l'altro processo
 - "numero di porta" che permette all'host ricevente di sapere a quale processo locale il messaggio è destinato

5

Livello Trasporto: cenni



- TCP (Transmission Control Protocol):
 - connection oriented
 - trasferimento affidabile
 - congestion control
 - nessuna garanzia su velocità di trasmissione e ritardo
- UDP (User Datagram Protocol):
 - connection less
 - trasferimento non affidabile
 - no congestion control
 - garanzie sul rispetto di una velocità minima (processi real-time)

6

Es.: Requisiti di alcune applicazioni



<u>Application</u>	<u>Data loss</u>	<u>Bandwidth</u>	<u>Time Sensitive</u>
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video: 10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
Instant messaging	no loss	elastic	yes and no

7

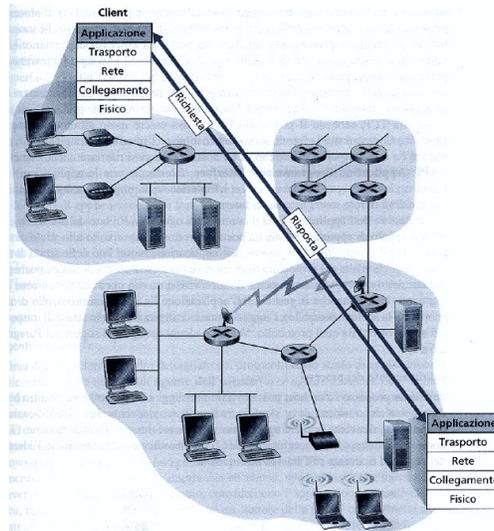
Es.: applicazioni e protocolli di trasporto



<u>Application</u>	<u>Application layer protocol</u>	<u>Underlying transport protocol</u>
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068,2616]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimediale	spesso proprietario (e.g. RealNetworks)	TCP or UDP
remote file server	NFS	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

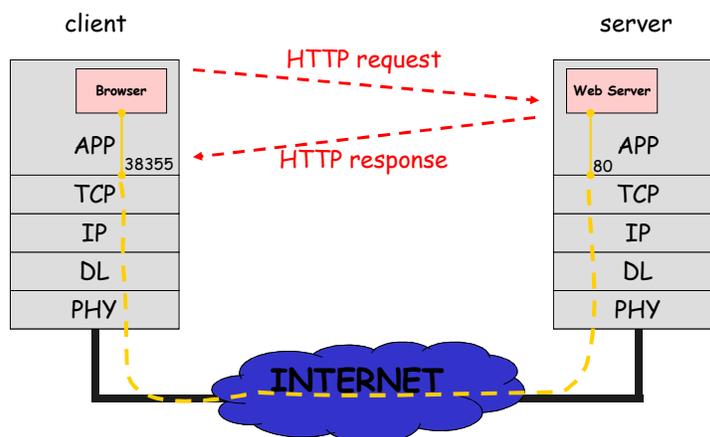
8

Paradigma Client/Server



9

Web: esempio di interazione Client→Server



10



HTTP

11



Il protocollo HTTP/1.0

- Si basa su TCP
- Il client apre un socket verso il porto TCP 80 del server (se non diversamente specificato)
- Il server accetta la connessione
- Il client manda una richiesta per uno specifico oggetto identificato mediante una URL
- Il server risponde e chiude la connessione
- **Il protocollo HTTP è stateless: né il server né il client mantengono a livello HTTP informazioni relative ai messaggi precedentemente scambiati**

12

URL : Uniform Resource Locator



- Un URL HTTP ha la seguente sintassi (RFC 2396) :
`http://host[:port]/path[#fragment][?query]`
- Host identifica il server
 - Può essere sia un nome simbolico che un indirizzo IP in notazione dotted decimal
- Port è opzionale; di default è 80
- Path identifica la risorsa sul server
 - es: images/sfondo.gif
- #fragment identifica un punto preciso all'interno di un oggetto
 - es: #paragrafo1
- ?query è usato per passare informazioni dal client al server
 - es: dati inseriti nei campi di una form

Es.: <http://www.unina.it/immatricolazioni.htm#ingegneria>

HTTP: versioni



- Il protocollo è definito in documenti RFC
- Negli anni si sono avute tre versioni:
 - http 0.9
 - http 1.0 (1996)
 - RFC 1945
 - http 1.1 (1997 e 1999)
 - RFC 2068
 - RFC 2616

NOTA: un RFC (Request For Comment) è un documento pubblico sottoposto alla comunità Internet al fine di essere valutato. Ciò che è un RFC rappresenta uno standard *de facto* nella comunità Internet. Tutti gli RFC possono essere reperiti al sito dell'Internet Engineering Task Force (<http://www.ietf.org>)

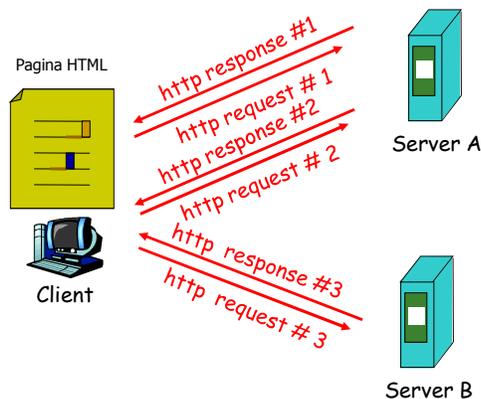
HTTP per il trasferimento di pagine web



- Tipicamente, una pagina web è descritta da un file testuale in formato HTML (*Hypertext Markup Language*)
- La pagina è identificata mediante un indirizzo, detto URL
- Un file HTML può contenere riferimenti ad altri oggetti che arricchiscono la pagina con elementi grafici
 - Es. sfondo, immagini, ecc.
- Ciascun oggetto è identificato dal proprio URL
- Questi oggetti possono trovarsi anche su server web diversi
- Una volta ricevuta la pagina HTML, il browser estrae i riferimenti agli altri oggetti che devono essere prelevati e li richiede attraverso una serie di connessioni HTTP

15

HTTP per il trasferimento di pagine web (2)



Es: richiesta di una pagina contenente immagini



1: il client apre una connessione TCP sul porto 80 verso l'indirizzo `www.unina.it`

2: il server è in ascolto sul porto 80 ed accetta la connessione

3: il client invia un messaggio di richiesta HTTP della home page

4: il server analizza la richiesta HTTP, prepara la risposta HTTP e la invia al client

5: il server chiude la connessione TCP

6: il client effettua il parsing del documento (es. HTML) contenuto nel messaggio di risposta HTTP, ne fa il rendering sullo schermo e rileva che all'interno della pagina sono presenti 3 collegamenti ad immagini.

7: per ciascuna delle immagini vengono ripetuti i passi da 1 a 5.

17

Il protocollo HTTP



- HTTP è un protocollo testuale
- I messaggi sono costituiti da sequenze di byte
- Ogni byte identifica un carattere secondo la tabella ASCII
- Il payload dei messaggi può essere comunque anche in formato binario (es. un'immagine GIF, un video, ecc.)

* Come si vedrà più avanti HTTP è utilizzato anche per altri scopi.

18

Il messaggio HTTP/1.0 request: un esempio



Un esempio di messaggio GET

```
GET /path/pagename.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: it

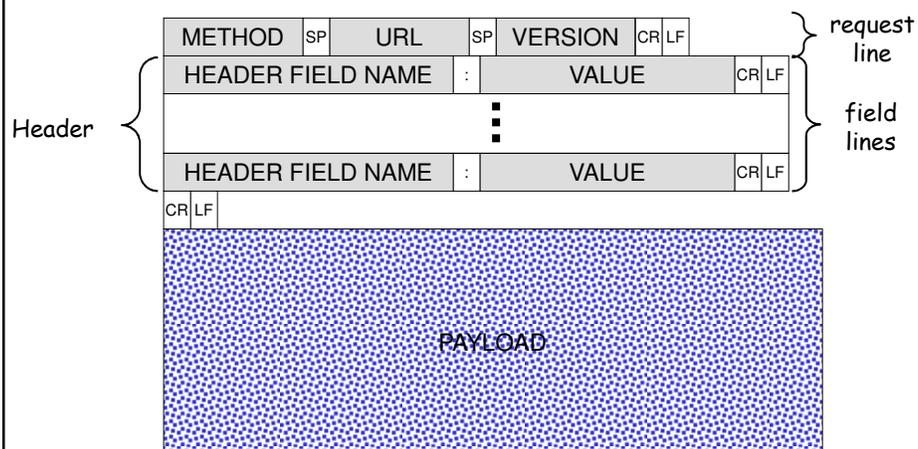
```

RIGA VUOTA

indica la fine del messaggio

19

Il messaggio HTTP/1.0 request



20

Il messaggio HTTP/1.0 response



codice di stato

HTTP/1.0 200 OK
Date: Mon, 16 Dec 2002 14:00:22 GMT
Server: Apache/1.3.24 (Win32)
Last-Modified: Fri, 13 Dec 2002 08:06:44 GMT
Content-Length: 222
Content-Type: text/html
RIGA VUOTA
PAYLOAD

21

Esempi di codici di stato



- 200 OK**
 - Successo: l'oggetto richiesto si trova più avanti nel messaggio
- 301 Moved Permanently**
 - L'oggetto richiesto è stato spostato.
 - Il nuovo indirizzo è specificato più avanti nel messaggio (Location:)
- 400 Bad Request**
 - Richiesta incomprensibile al server
- 404 Not Found**
 - Il documento non è presente sul server
- 505 HTTP Version Not Supported**
 - La versione del protocollo HTTP usata non è supportata dal server

22

Lo scambio di messaggi



- Il metodo **GET**
- Il metodo **HEAD**
- Il metodo **POST**
- Il metodo **PUT**

- *La response*

23

Il metodo GET



- Uno dei più importanti metodi di HTTP è **GET**
- Usato per richiedere una risorsa ad un server
- Questo è il metodo più frequente, ed è quello che viene attivato facendo click su un link ipertestuale di un documento HTML, o specificando un URL nell'apposito campo di un browser
- GET può essere:
 - **assoluto**
 - la risorsa viene richiesta senza altre specificazioni
 - **condizionale**
 - si richiede la risorsa se è soddisfatto un criterio indicato negli header **If-match**, **If-modified-since**, **If-range**, ecc.
 - **parziale**
 - si richiede una sottoparte di una risorsa memorizzata

24

Il metodo GET: un esempio

The screenshot displays a network capture in Wireshark. The packet list pane shows a series of packets, with packet 5 highlighted as a GET request. The packet details pane shows the following information:

- Frame 5 (627 bytes on wire, 627 bytes captured)
- Ethernet II, Src: Wlstron_2e:cb:5b (00:0a:e4:2e:cb:5b), Dst: Ciscoc_6d:04:00 (00:14:fl:6d:04:00)
- Internet Protocol, Src: 143.225.229.163 (143.225.229.163), Dst: 209.85.129.147 (209.85.129.147)
- Transmission Control Protocol, Src Port: 4150 (4150), Dst Port: http (80), Seq: 1, Ack: 1, Len: 373
- Hypertext Transfer Protocol
 - Host: www.google.com/r/n
 - User-Agent: Mozilla/5.0 (Windows; u; windows NT 5.1; it-IT; rv:1.7.12) Gecko/20050919 Firefox/1.0.7/r/n
 - Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5/r/n
 - Accept-Language: it, it-IT;q=0.8,en-us;q=0.5,en;q=0.3/r/n
 - Accept-Encoding: gzip,deflate/r/n
 - Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7/r/n
 - Keep-Alive: 300/r/n
 - Connection: keep-alive/r/n
 - Cookie: PREF=ID=42ee8329038ae07:FF=4:LD=en:NR=10:CR=2:TM=1169647218:LM=1187706763:GM=1:S=zCPXwUB24XF7Aytk; S=awfe=QD2p-kks/r/n

The packet bytes pane shows the raw data of the request, including the host and user-agent strings.

Il metodo HEAD

- Simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi, senza il corpo
- Usato per verificare:
 - **la validità di un URI**
 - la risorsa esiste e non è di lunghezza zero
 - **l'accessibilità di un URI**
 - la risorsa è accessibile presso il server, e non sono richieste procedure di autenticazione del documento
 - **la coerenza di cache di un URI**
 - la risorsa non è stata modificata nel frattempo, non ha cambiato lunghezza, valore hash o data di modifica

Il metodo POST



- Il metodo **POST** serve per trasmettere delle informazioni dal client al server, ma senza la creazione di una nuova risorsa
- POST viene usato per esempio per sottomettere i dati di una form HTML ad un'applicazione sul server
- I dati vengono trasmessi nel body della richiesta
- Il server può rispondere positivamente in tre modi:
 - **200 Ok**: dati ricevuti e sottomessi alla risorsa specificata; è stata data risposta
 - **201 Created**: dati ricevuti, la risorsa non esisteva ed è stata creata
 - **204 No content**: dati ricevuti e sottomessi alla risorsa specificata; non è stata data risposta
- Non è l'unico modo per inviare dati

27

Il metodo POST: un esempio



The screenshot shows a network capture in Wireshark. The top pane shows a list of packets. Packet 6 is highlighted in red, indicating it is selected. The middle pane shows the details of the selected packet, which is an HTTP POST request. The bottom pane shows the raw bytes of the packet, which are hex and ASCII encoded.

No.	Time	Source	Destination	Protocol	Info
6	0.107593	192.168.1.2	212.52.84.18	HTTP	POST /ema...l.php HTTP/1.1
14	5.406089	212.52.84.18	192.168.1.2	HTTP	200 OK (text/html)
16	5.426579	192.168.1.2	212.52.84.18	HTTP	GET /favIcon.ico HTTP/1.1
19	5.502876	212.52.84.18	192.168.1.2	HTTP	200 OK (image/x-icon)
23	5.519777	192.168.1.2	212.52.84.18	HTTP	GET /css/online... HTTP/1.1
25	5.611096	212.52.84.18	192.168.1.2	HTTP	200 OK (text/html)
33	5.836637	192.168.1.2	199.210.87.1	HTTP	GET /cgi-bin/adv_getbanner.js?site=Mail&ip=liberobqstarget HTTP/1.1
38	5.951878	199.210.87.1	192.168.1.2	HTTP	200 OK (text/javascript)
41	5.968380	192.168.1.2	212.52.84.18	HTTP	GET /xan/PC/comp... HTTP/1.1
42	5.968054	192.168.1.2	212.52.84.18	HTTP	GET /xan/PC/comp... HTTP/1.1

Frame 6 (881 bytes on wire, 881 bytes captured)
Ethernet II, Src: Wlstron_2e:cb:5b (00:0a:e4:2e:cb:5b), Dst: D-Link_1e:6a:d1 (00:17:9a:1e:6a:d1)
Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 212.52.84.18 (212.52.84.18)
Transmission Control Protocol, Src Port: 2839 (2839), Dst Port: http (80), Seq: 1, Ack: 1, Len: 827
Hypertext Transfer Protocol
POST /ema...l.php HTTP/1.1
Host: wpp8.1libero.it
User-Agent: Mozilla/5.0 (Windows; u; windows NT 5.1; it-IT; rv:1.7.12) Gecko/20050919 Firefox/1.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.libero.it
Cookie: ...utmz=267072147.1190633699.2.3.utmcsr=HP%2BLibero|utmccn=Infostrada%2GADSL|utmcnd=Lancio1; LiberoMail=2dccc1ab3e03

HTTP Referer (http.referer), 32 bytes

28

Il metodo PUT



- Il metodo **PUT** serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata
 - Esempio: upload di un file
- In generale, l'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET in seguito con lo stesso nome

29

HTTP: Response



- La risposta HTTP è un messaggio testuale formato da una riga iniziale, da header facoltativi ed eventualmente un body (corpo)

Version status-code reason-phrase CRLF } request line

[Header] } Header
[Header] }

CRLF

Body

dove:

[Header] indica un elemento opzionale

CRLF indica la sequenza di caratteri di codice ASCII

$0D_{16} = 13_{10} \rightarrow$ CR = Carriage Return

$0A_{16} = 10_{10} \rightarrow$ LF = Line Feed

30

HTTP: Response (2)



- Un Esempio:

HTTP/1.1 200 OK

Date: Thu, 10 Apr 2003 11:46:53 GMT

Server: Apache/1.3.26 (Unix) PHP/4.0.3pl1

Last-Modified: Wed, 18 Dec 2002 12:55:37 GMT

Accept-Ranges: bytes

Content-Length: 7394

Content-Type: text/html

<HTML> ... </HTML>

31

HTTP Response: un esempio



(Untitled) - Ethereal

File Edit View Go Capture Analyze Statistics Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452
2	3.270549	143.225.229.163	209.85.129.147	TCP	4150 > http [SYN] Seq=0 Len=0 MSS=1460
3	3.304386	209.85.129.147	143.225.229.163	TCP	http > 4150 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
4	3.304443	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORR]
5	3.308079	143.225.229.163	209.85.129.147	HTTP	GET / HTTP/1.1
6	3.402851	209.85.129.147	143.225.229.163	TCP	http > 4150 [ACK] Seq=1 Ack=574 Win=6611 Len=0
7	3.413274	209.85.129.147	143.225.229.163	TCP	[TCP segment reassembled PDU]
8	3.415291	209.85.129.147	143.225.229.163	HTTP	HTTP/1.1 200 OK (text/html)
9	3.433314	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=574 Ack=2344 Win=65535 [TCP CHECKSUM II]
10	6.038651	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452

Frame 8 (1167 bytes on wire (3167 bytes captured))

Ethernet II, Src: Cisco_Gd:04:00 (00:14:f1:6d:04:00), Dst: Wlstron_2e:cb:5b (00:0a:e4:2e:cb:5b)

Internet Protocol, Src: 209.85.129.147 (209.85.129.147), Dst: 143.225.229.163 (143.225.229.163)

Transmission Control Protocol, Src Port: http (80), Dst Port: http (80), Seq: 1431, Ack: 574, Len: 1113

[Reassembled TCP Segments (2543 bytes): #7(1430), #8(1113)]

Hypertext Transfer Protocol

- HTTP/1.1 200 OK(V)
- cache-control: private(V)
- content-type: text/html; charset=UTF-8(V)
- content-encoding: gzip(V)
- server: gws(V)
- content-length: 2364(V)
- date: Mon, 24 Sep 2007 10:25:21 GMT(V)
- Content-Encoded entity body (gzip): 2364 bytes -> 5472 bytes
- Line-based text data: text/html

0000 00 0a e4 2e cb 5b 00 14 f1 6d 04 00 08 00 45 00[. .m...E.
0010 04 31 b6 c6 00 00 33 06 04 43 d1 55 81 95 8f e13...C.U....
0020 e5 a3 00 50 10 36 a6 a2 e3 00 f7 46 f9 c0 50 18P.6...F.P.
0030 19 d3 82 fa 00 00 d4 67 d5 49 d0 32 35 8c 5e 2fg...I.25.A/
0040 e4 85 45 be 74 74 be 84 3f a4 93 b3 03 79 5c ffE.t...?..Y\
0050 21 58 ee d9 69 50 8a 9e a6 39 99 0c c9 89 87 88x..P...S.....
0060 69 05 41 a1 28 13 28 10 20 56 74 ee d3 bf 96 c3 1..A..C..t...
0070 ea 0c d5 36 7a ea a5 85 7a 7c d8 40 ef 85 0e a5 ..A...e>A

Frame (1167 bytes) [Reassembled TCP (2543 bytes)] [Uncompressed entity body (5472 bytes)]

File: "C:\Documents\1146180505\1170001\ethereal\20070924\3676.pcap" [9-10-D-10-M-0-Proc-0]

32

Status code



- Lo status code è un numero di tre cifre, di cui la prima indica la classe della risposta, e le altre due la risposta specifica
- Esistono le seguenti classi:
 - **1xx: Informational**
 - Una risposta temporanea alla richiesta, durante il suo svolgimento
 - **2xx: Successful**
 - Il server ha ricevuto, capito e accettato la richiesta
 - **3xx: Redirection**
 - Il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client per portare a termine la richiesta
 - **4xx: Client error**
 - La richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata)
 - **5xx: Server error**
 - La richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema interno

33

Status code: alcuni esempi



- **100 Continue**
 - se il client non ha ancora mandato il body
- **200 Ok**
 - GET con successo
- **201 Created**
 - PUT con successo
- **301 Moved permanently**
 - URL non valida, il server conosce la nuova posizione
- **400 Bad request**
 - errore sintattico nella richiesta
- **401 Unauthorized**
 - manca l'autorizzazione
- **403 Forbidden**
 - richiesta non autorizzabile
- **404 Not found**
 - URL errato
- **500 Internal server error**
 - tipicamente un programma in esecuzione sul server ha generato errore
- **501 Not implemented**
 - metodo non conosciuto dal server

34