HTTP szolgáltatások

A HTTP alapjai

A HTTP (HyperText Transfer Protocol) egy alkalmazás réteg protokoll, amely kliens–szerver modellben működik. Elsősorban webes erőforrások (HTML, CSS, JS, képek, API válaszok stb.) továbbítására használják.

Alapvető jellemzők:

- Állapotmentes (stateless)
- Kérés-válasz alapú (request–response)
- TCP (HTTP/1.x, HTTP/2) vagy QUIC/UDP (HTTP/3) felett működik
- Szöveges (HTTP/1.x) illetve bináris (HTTP/2, HTTP/3)

HTTP működési modell

A kommunikáció menete:

1. Kliens kapcsolatot nyit a szerverhez
2. Kliens HTTP kérést küld
3. Szerver HTTP választ küld
4. Szerver lezárja a kapcsolatot (HTTP 0.9, HTTP 1.0) vagy nyitva marad ha szükséges (HTTP 1.1, HTTP 2)

HTTP protokol verziók

HTTP/0.9 (1991)

A HTTP/0.9 volt a legelső, kísérleti változata a protokollnak, amelyet Tim Berners-Lee fejlesztett ki a CERN-ben a World Wide Web kezdeti időszakában.

Jellemzői:

- Rendkívül egyszerű
- Csak egyetlen metódus létezett: GET
- Nem léteztek HTTP fejlécek
- Nem volt státuszkód
- Nem volt verziószám a kérésben

A válasz kizárólag HTML tartalom lehet.

Kérés

```
# telnet szervernév 80
GET /index.html
```

Válasz

A szerver egyszerűen visszaküldi a HTML dokumentumot:

```
<html>
<body>Hello World</body>
</html>
```

A válaszban nincs:

- állapot sor
- fejléc mező
    - Content-Type
    - Content-Length

Korlátok

- Nem támogatta a képeket, CSS-t, külön erőforrásokat
- Nem volt hibakezelési mechanizmus
- Nem alkalmas komplex webalkalmazásokhoz

Összegzés

A HTTP/0.9 egy rendkívül egyszerű, csak GET-et támogató, fejléc nélküli protokoll volt, amely kizárólag HTML dokumentum visszaküldésére szolgált. Ez tekinthető a modern webkommunikáció kiindulópontjának.

HTTP/1.0 (1996)

Jellemzők:

- Minden kérés külön TCP kapcsolat
- Nincs alapértelmezett persistent connection
- Egyszerű szöveges protokoll
- Nincs kötelező Host fejléc

Problémák:

- Sok TCP handshake

- Hosszú késleltetés
- Nem hatékony párhuzamos erőforrás-letöltés

Példa kommunikáció:

Egyszerű kérés felépítése:

```
$ telnet server 80
Trying N.N.N.N...
Connected to szerver.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Sun, 15 Feb 2026 03:32:17 GMT
Server: Apache/2
Last-Modified: Tue, 27 Feb 2024 06:32:03 GMT
ETag: "0-612572f2e4ba2"
Accept-Ranges: bytes
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8

Connection closed by foreign host.
```

HTTP/1.1 (1997)

Fejlesztések:

- Persistent connection (Keep-Alive)
- Host header kötelező
- Chunked transfer encoding
- Pipeline támogatás

Példa a hagyományos kapcsolatra:

```
$ telnet server 80
Trying N.N.N.N...
Connected to server.
Escape character is '^]'.
GET / HTTP/1.1
Host: server

HTTP/1.1 403 Forbidden
Date: Sun, 15 Feb 2026 03:47:36 GMT
Server: Apache/2.4.62 (AlmaLinux)
Last-Modified: Mon, 24 Mar 2025 16:15:24 GMT
ETag: "1680-63118e9567f00"
Accept-Ranges: bytes
```

```
Content-Length: 130
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<body>
Hello world!
</body>
</html>
Connection closed by foreign host.
```

Példa persistent kapcsolatra:

```
$ telnet server 80
Trying N.N.N.N...
Connected to server.
Escape character is '^]'.
GET / HTTP/1.1
Host: xxx
Connection: keep-alive

TARTALOM
GET / HTTP/1.1
Host: xxx
Connection: close

TARTALOM

Connection closed by foreign host.
```

Előny:

- Kevesebb TCP kapcsolat
- Jobb teljesítmény

Probléma:

HOL (Head-of-line) blokkolás (több kérés esetén). Mivel a kérések sorrendjében válaszol a szerver, ezért egy lassú feldolgozási művelet blokkolja a többi kérés kiszolgálását.

HTTP/2 (2015)

Alapja: a Google 2009-ben fejlesztett SPDY (kiejtve: “speedy”) kísérleti alkalmazásrétegű protokoll a HTTP/1.1 teljesítmény korlátainak megszüntetésére.

Fő jellemzők:

- Bináris protokoll
- Multiplexing (több stream egy TCP kapcsolaton)
- Header compression (HPACK)

- Server Push

Előny:

- Megszűnik az alkalmazás-szintű head-of-line blocking
- Jelentősen jobb teljesítmény

Hátrány:

- TCP head-of-line blocking továbbra is fennáll csomagszinten

Példa:

```
$ curl -v -s -o /dev/null --http2 https://server
* Host server:443 was resolved.
* IPv6: (none)
* IPv4: N.N.N.N
*   Trying N.N.N.N:443...
* ALPN: curl offers h2,http/1.1
} [5 bytes data]
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
} [1565 bytes data]
*  CAfile: /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem
*  CApath: none
{ [5 bytes data]
* TLSv1.3 (IN), TLS handshake, Server hello (2):
{ [122 bytes data]
* TLSv1.3 (IN), TLS change cipher, Change cipher spec (1):
{ [1 bytes data]
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
{ [25 bytes data]
* TLSv1.3 (IN), TLS handshake, Certificate (11):
{ [2577 bytes data]
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
{ [264 bytes data]
* TLSv1.3 (IN), TLS handshake, Finished (20):
{ [52 bytes data]
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
} [1 bytes data]
* TLSv1.3 (OUT), TLS handshake, Finished (20):
} [52 bytes data]
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384 / x25519 / RSASSA-
PSS
* ALPN: server accepted http/1.1
* Server certificate:
*  subject: CN=server
*  start date: Jan 20 03:10:26 2026 GMT
*  expire date: Apr 20 03:10:25 2026 GMT
*  subjectAltName: host "server" matched cert's "server"
*  issuer: C=US; O=Let's Encrypt; CN=R12
*  SSL certificate verify ok.
*   Certificate level 0: Public key type RSA (2048/112 Bits/secBits), signed
```

```
using sha256WithRSAEncryption
*   Certificate level 1: Public key type RSA (2048/112 Bits/secBits), signed
using sha256WithRSAEncryption
*   Certificate level 2: Public key type RSA (4096/152 Bits/secBits), signed
using sha256WithRSAEncryption
* Connected to r-l.hu (N.N.N.N) port 443
* using HTTP/1.x
} [5 bytes data]
> GET / HTTP/1.1
> Host: r-l.hu
> User-Agent: curl/8.15.0
> Accept: */*
>
* Request completely sent off
{ [5 bytes data]
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
{ [281 bytes data]
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
{ [281 bytes data]
< HTTP/1.1 200 OK
< Date: Sun, 15 Feb 2026 04:10:24 GMT
< Server: Apache/2.4.62 (AlmaLinux)
< Last-Modified: Sun, 15 Feb 2026 04:09:05 GMT
< ETag: "0-64ad4ffd21f44"
< Accept-Ranges: bytes
< Content-Length: 0
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host r-l.hu left intact
```

Több tartalom lekérése:

```
$ curl --http2 -s -o /dev/null https://server/index.html
https://server/style.css https://server/app.js
```

HTTP/3 (2022)

Alapja: A QUIC (Quick UDP Internet Connections) egy modern, titkosított, multiplexelt transzport protokoll, amely UDP felett működik, és amelyet eredetileg a Google fejlesztett ki. IETF szabvány (RFC 9000).

A QUIC célja a TCP + TLS + HTTP/2 kombináció leváltása egy egyetlen, hatékonyabb protokollal, amely csökkenti a késleltetést és javítja a kapcsolat stabilitását.

Jellemzők:

- TCP helyett UDP
- TLS 1.3 integrált
- Nincs TCP head-of-line blokkolódás
- Gyorsabb kapcsolatfelépítés 0-RTT (Round-Trip Time)

- Connection ID-t használ a TCP IP + port azonosítás helyett

Előny:

- Jelentősen alacsonyabb késleltetés
- Mobil hálózatokon stabilabb

HTTP verziók összehasonlítása

Verzió	Szállítási réteg	Formátum	Multiplexing	Head-of-line blocking
HTTP/1.0	TCP	Szöveges	Nem	Igen
HTTP/1.1	TCP	Szöveges	Korlátozott (pipeline)	Igen
HTTP/2	TCP	Bináris	Igen	TCP szinten igen
HTTP/3	QUIC (UDP)	Bináris	Igen	Nem

HTTP kérés metódusok

HTTP metódusok összefoglaló táblázat

A metódus biztonságos (safe), ha nem változtatja meg a szerver állapotát. Ilyen metódusok

Egy metódus idempotens, ha ugyanazt a kérést többször elküldve a szerver állapota az első végrehajtás után már nem változik tovább.

Metódus	Safe	Idempotens	Van body?	Tipikus használat
GET	Igen	Igen	Nem	Erőforrás lekérdezése
HEAD	Igen	Igen	Nem	Csak header lekérdezése
POST	Nem	Nem	Igen	Adatküldés, űrlap, API
PUT	Nem	Igen	Igen	Erőforrás létrehozása/felülírása
DELETE	Nem	Igen	Nem	Erőforrás törlése
OPTIONS	Igen	Igen	Nem	Támogatott metódusok lekérdezése
PATCH	Nem	Nem	Igen	Részleges módosítás
TRACE	Igen	Igen	Nem	Diagnosztika

Példák:

Kapcsolódás minden esetben:

```
telnet server 80
```

Fontos:

- HTTP/1.1 esetén kötelező a `Host` header
- A header részt üres sor zárja le
- Body esetén kötelező a `Content-Length`

GET

```
GET / HTTP/1.1
Host: server
Connection: close
```

HEAD

```
HEAD / HTTP/1.1
Host: server
Connection: close
```

POST

```
POST /login HTTP/1.1
Host: server
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Connection: close

username=test&password=123
```

PUT

```
PUT /file.txt HTTP/1.1
Host: server
Content-Type: text/plain
Content-Length: 11
Connection: close

Hello World
```

DELETE

```
DELETE /file.txt HTTP/1.1
Host: server
Connection: close
```

OPTIONS

```
OPTIONS / HTTP/1.1
Host: server
```

```
Connection: close
```

Ez utóbbi kérésre a válasz:

```
...
Allow: GET, POST, HEAD
...
```

PATCH

```
PATCH /user/1 HTTP/1.1
Host: server
Content-Type: application/json
Content-Length: 18
Connection: close

{"name":"ujnev"}
```

TRACE

```
TRACE / HTTP/1.1
Host: server
Connection: close
```

Fontos a sorrend!

- A metódus az első szó a kérésben
- A státuszkód a válasz első sorában jelenik meg
- A header és a body között üres sor található
- A kapcsolat lezárását a `Connection: close` header vagy a szerver TCP bontása jelzi

HTTP válasz (állapot) kódok

Tartomány	**Jelentés**
1xx	Információ
2xx	Siker
3xx	Átirányítás
4xx	Kliens hiba
5xx	Szerver hiba

Kód	**Megnevezés**	**Jelentés**
100	Continue	A kliens folytathatja a kérést
101	Switching Protocols	Protokollváltás (pl. WebSocket)
102	Processing	A kérés feldolgozás alatt

200	OK	Sikeres kérés
201	Created	Erőforrás létrejött
202	Accepted	Feldolgozás elfogadva

203	Non-Authoritative Information	Nem hiteles forrásból származó válasz
204	No Content	Sikeres, de nincs válasz body
205	Reset Content	Kliens nézetének visszaállítása
206	Partial Content	Részleges tartalom (Range kérés)

300	Multiple Choices	Több válaszlehetőség
301	Moved Permanently	Végleges átirányítás
302	Found	Ideiglenes átirányítás
303	See Other	Más erőforrás GET-tel
304	Not Modified	Nem változott (cache)
307	Temporary Redirect	Ideiglenes, metódust megőrzi
308	Permanent Redirect	Végleges, metódust megőrzi

400	Bad Request	Hibás kérés
401	Unauthorized	Hitelesítés szükséges
402	Payment Required	Fenntartott
403	Forbidden	Tiltott hozzáférés
404	Not Found	Nem található
405	Method Not Allowed	Metódus nem engedélyezett
406	Not Acceptable	Nem elfogadható formátum
407	Proxy Authentication Required	Proxy hitelesítés szükséges
408	Request Timeout	Kérés időtúllépés
409	Conflict	Ütközés
410	Gone	Végleg eltűnt
411	Length Required	Content-Length hiányzik
412	Precondition Failed	Előfeltétel nem teljesült
413	Payload Too Large	Túl nagy kérés
414	URI Too Long	Túl hosszú URI
415	Unsupported Media Type	Nem támogatott médiatípus
416	Range Not Satisfiable	Érvénytelen tartomány
417	Expectation Failed	Expect header hiba
418	I'm a teapot	RFC humoros státuszkód
429	Too Many Requests	Túl sok kérés (rate limit)

500	Internal Server Error	Belső szerverhiba
501	Not Implemented	Nem implementált
502	Bad Gateway	Hibás gateway válasz
503	Service Unavailable	Szolgáltatás nem elérhető
504	Gateway Timeout	Gateway időtúllépés
505	HTTP Version Not Supported	HTTP verzió nem támogatott

HTTPS

A HTTPS nem külön protokoll, hanem:

HTTP + TLS titkosítás

Alapértelmezett port:

80 → HTTP

443 → HTTPS

Telnet helyett HTTPS teszteléshez:

```
openssl s_client -connect example.com:443
```

Összegzés

HTTP/1.x: egyszerű, szöveges, TCP-alapú

HTTP/2: bináris, multiplexelt, gyorsabb

HTTP/3: QUIC-alapú, modern, alacsony latency

Telnet kiváló eszköz a HTTP/1.x működés szemléltetésére