

WEB01-http

Statikus kiszolgálás	Dinamikus kiszolgálás
Egyszerű, olcsó, hatékony	Bonyolult, Drága, Lassabb
A tartalom csak a szerveren található fájlok manipulációjával módosítható	A tartalom újraindítás nélkül és telepítés nélkül frissíthető.
!= Statikus weboldal! : Statikus kódból módosíthatjuk a JS tartalmát.	Dinamikus weboldal != Dinamikus kiszolgálás! Léteznek single page application-ök, amik egy API-ról töltik le az adatokat.

Session: Egy felhasználó első és utolsó kérése közt lezajló kérés-válasz tranzakciók.

User agent: A kliens általános megnevezése, bármilyen alkalmazás, amely HTTP kérést tud kiszolgálni.

- **User agent sniffing:** más tartalom megjelenítése különböző klienseknek.
- **User agent spoofing:** a User-Agent mező meghamisítása.

Kérés és válasz elemei

GET: Erőforrás letöltése

POST: Adat felküldése a szerverre a kérés body részében.

PUT, PATCH: Frissíti a megadott erőforrást a szerveren.

DELETE: Töröl egy megadott erőforrást.

HEAD: HTTP fejléc lekérdezése a megadott erőforrásról.

OPTIONS: Visszaadja a szerver által támogatott HTTP metódusok listáját.

TRACE: visszaküldi a kapott kérést. (Nem mindig támogatott, csak tesztelésre szokás használni)

Biztonságos metódusok

Csak információ **letöltésére** szolgálnak. Nincs mellékhatásuk, nem változtatnak állapotot a szerveren. A kliens gond nélkül megpróbálhatja újrafuttatni őket.

- Ezek a **GET, HEAD, OPTIONS, TRACE**

Idempotens metódus

Többszöri végrehajtása ugyanazt a hatást váltja ki, mint az egyszeri.

Minden biztonságos metódus egyben idempotens is!

- Továbbá ezek még a **PUT, DELETE** metódusok
- A **DELETE** nem dobhat hibát, hogy nem található az erőforrás.
 - Lehet, hogy más státuszkóddal tér vissza többszöri futtatásra a DELETE, de az idempotens metódusok lényege az, hogy a többszöri futtatástól a szerver **belső állapota** nem fog változni
- A **POST** általában nem idempotens. A **PATCH** sem, mert a részleges frissítés megvalósítása lehet olyan, hogy mellékhatása van.

Válasz

A beérkező kérést a webszerver feldolgozza és előállítja a **szöveges HTTP válaszüzenetet**.

URL

Meghatározza az erőforrás elérését is.



Általános formátum: `protocol://username:password@FQDN:port/path/file?variable1=value1&variable2=value2#fragment`

Abszolút URL: mindentől függetlenül, egyértelműen határozza meg az útvonalat.

Relatív URL: az aktuális dokumentumhoz vagy a szerver gyökeréhez képest relatív útvonal.

🔗 Mi töltődik be a `http://www.google.com@www.bme.hu` hatására?

`www.bme.hu`, ahova a "`www.google.com`" felhasználónévvel próbálunk majd meg belépni.



Pár státuszkód

1xx - Information

- **100:** Continue
- **101:** Switching protocols

2xx - Success

- **200:** OK
- **201:** Created
- **204:** No content

3xx - Redirect

- **301:** Moved permanently
- **304:** Not modified

4xx - Client Error

- **400:** Bad request
- **401:** Unauthorized
- **403:** Forbidden
- **404:** Not found
- **405:** Method not allowed
- **410:** Gone
- **413:** Request entity too large
- **414:** Request URI too long

5xx - Server Error

- **500**: Internal server error
- **503**: Service unavailable

Kérések jelentései általában REST APIban:

GET: Lekérdezi az URL-en található erőforrást.

POST: Létrehoz egy új erőforrást a megadott URL-en.

PUT: Létrehozza, vagy lecseréli az adott URL-en levő erőforrást.

PATCH: Részlegesen frissíti az erőforrást.

DELETE: Törli az URL-en levő erőforrást.

Fejlécmezők

```
// Gyorsítótárral kapcsolatos mezők
Cache-Control: no-cache
Expires: dátum
If-Modified-Since: dátum
Last-Modified: dátum
ETag: verzió

//Biztonsággal kapcsolatos mezők
Authorization: BasicTXlEb21haW5cTXlDb21wdXRlcjpdXB1c1NlY3JldFBhc3N3b3Jk
WWW-Authenticate: Basic realm="MyComputer"
X-Frame-Options: SAMEORIGIN
DNT: 1
```

HTML

Jelölőnyelv, ami leírja a böngészőnek, hogyan épül fel a weboldal struktúrája.

Attribútumok: Extra információt adnak az elemhez. (Egyedi azonosító, név, CSS osztályok..)

- A nyitó tagbe írjuk be, a neve után = jel következik. Bool értéknél elhagyható az érték, elég kiírni az attrib. nevét.
- Megadható javascript függvény is, amit adott eseményre meg akarunk hívni. pl. `onclick=randomFunction(this)`

<head>

- Itt adhatunk meg az oldalra vonatkozó metaadatokat (karakterkódolás, oldal címe, cache beállítások, stb.)
- Itt hivatkozhatunk css fájlokra is

Szemantikus oldalváz

Több különböző tag, pl.: <header>, <nav>, <aside>, <section>, <article>, <footer>, stb..

- **Jobb**, mert egyes elemek jelentéssel bírnak.
- A böngészők, keresőmotorok szeretik, értelmezik

Blokk elemek	Inline elemek
Mindig új sorban jelennek meg.	Ugyanabban a sorban jeleníti meg, mint amiben az előtte levő elem van.
Csak blokkszintű elembe lehet beágyazni.	Blokkszintű elem tartalmának egy része.
pl: div , p , form	pl: span , a , img , strong

Táblázatok

```
<table>
  <caption>Csoport lista</caption>
  <thead>
    <tr>
      <th scope="col">Név</th>
      <th scope="col">Életkor</th>
      <th scope="col">Jel</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">Gábor</th>
      <td>39</td>
      <td>zászló</td>
    </tr>
  </tbody>
</table>
```

Űrlapok

```

<form action="register.aspx" method="get">
  <label for="name">Név:</label>
  <input type="text" id="name"
    name="name" value="Gincsei Gábor">
  <br>
  <input type="submit" value="Küldés">
</form>

```

Input típusok

Egyszerű bevitel: text , password, number, textarea

Választós mező: radio, checkbox

Gomb (type)-ok: button, submit, reset

Fájl: File

- Feltöltés:
 - Olyan input, amivel fájlt lehet kiválasztani: `<input type="file">`
 - Formon (enctype), vagy inputon a formenctype megadása
 - `<input formenctype="multipart/form-data">` VAGY
 - `<form ... enctype="multipart/form-data">`
 - Több fájl feltöltéséhez multiple attribútum:
 - `<input type="file" multiple>`

Dátum típusok: date, datetime, datetime-local, month, time, week

Egyéb: email, range, search, tel, url, color

Legördülő lista: select

- optgroup*: választható elemek csoportja
- option*: konkrét választható elem:

```

<select>
  <option value="alma">Alma</option>
  <option value="korte">Körte</option>
</select>

```

Szűrhető lista: datalist

- Select helyett datalist, de ugyanúgy option-ök vannak.
- Annyival tud többet, hogy lehet hozzá készíteni egy keresőmezőt, amivel lehet a listán belüli elemekben keresni:

```
<input list="listaID">
```

Beviteli mezőhöz tartozó címke: <label>

- for attribútum: megadhatjuk, hogy melyik inputhoz tartozik.

Input attribútumok

placeholder: Helyőrző

readonly: Csak olvasható.

hidden: Nem látható közvetlenül. (Nyilván webvizsgálóban igen.)

disabled: Letiltott. **Nem kerül felküldésre a szerver felé!**

autocomplete="off": Korábbi értékek felajánlásának kikapcsolása.

autofocus: Automatikusan kapjon fókuszt.

Validáció

required: Kötelezően kitöltendő mező

pattern: Reguláris kifejezés, amire illeszkednie kell a bevitt adatoknak.

maxlength: maximális hossz

min, *max*, *step* = min/max érték, és lépésköz

Javascript kódból is validálhatunk (meg jQueryből is).

Állapotmegőrzés

A HTTP alapból állapotmentes. Ez annyira nem jó akkor, amikor szeretnénk, hogy megjegyezze az oldal a belépési adatainkat (pl Gmail), vagy a kosarunk tartalmát egy webshopban.

Megoldási lehetőségek:

kliens->szerver

1. Session információ küldése az adattal együtt, minden kérésnél

- + Nem igényel a szerveroldalon erőforrást, sok felhasználót ki lehet vele egyszerre szolgálni.
 - + A tárolható adatok mérete korlátozott (Nem skálázódik jól nagy adatmennyiségre)
 - Az adatok mindig utaznak a hálózaton, pazaroljuk a sávszélességet.
 - Biztonsági kérdések szempontjából sem annyira előnyös.
 - Ha valaki belelát a forgalomba, akkor a session adatokkal kiadhatja magát másnak (*man-in-the-middle*), módosíthatja az adatot nem kívánt módon: *tampering*. Megoldás: **digitális aláírás**
 - vagy csak szimplán beleláthat (*eavesdropping*): Megoldás: **HTTPS**
- Megvalósítás:*
- URL mezőben: a végére illesztve egy `?` után. Ha több paramétert is el akarunk küldeni, akkor köztük `&` jelet írunk: www.google.com?page=2&q=valami
 - Rejtett form mező: `<input type="hidden" name="id" value="2">`

2. Cookie-ban (ez kicsit még az 1. megvalósításhoz tartozik.)

- "memória a HTTP-hez"
- **Session cookie**: Csak a munkamenet idejére létezik, a böngésző bezárásával törlődik.
- **Permanent cookie**: Diszkre mentődik.
- tartalma:
 - *Name*: a süti neve.
 - *Value*: csak string lehet
 - *Expiration date*: lejáratási idő
 - *Path*: URL-ben minek kell szerepelnie, hogy elküldje a böngésző
 - *Domain*: melyik hostokra kell küldeni (alapértelmezetten oda, ahonnan letöltöttük az oldalt)
 - *Secure*: Csak HTTPS-en használható-e
 - *HttpOnly*: Állítható vele, hogy lehessen-e JavaScriptből módosítani.
- A süti törlésére nincs külön fejléc. Felül lehet írni üres tartalommal és egy már lejárt lejáratási dátummal.
- A böngésző minden alkalommal visszaküldi a szerver felé, ha egyezik a domain és path
- **Nyílt szöveggént utazik**
- **Változtatható a tartalma**
- **Nem garantálható az eredete**
- **Script is hozzáférhet és módosíthatja**
- Ezeknek nagy részéhez létezik valamilyen védelem (titkosítás, aláírás)

3. Local és Session Storage

4. IndexedDB

5. FileSystem (csak virtuális fájlrendszer)

kliens + szerver

1. Az információk a szerveren tárolódnak, csak a munkamenet azonosítója utazik a kliens + szerver közt.
Minden előny, ami a másinál hátrány volt, de **sok memóriát igényel**

HTTPS

A HTTP alaptól titkosítatlan, de TLS vagy SSL-el már titkosított (HTTPS)

Ez nem önálló protokoll! TLS, SSL felett már protokoll is utazhat.

Port: 443

Funkciói:

- Szerver azonosítása: *pontosan* kivel kommunikál a gép
- Kommunikáció titkosítása: *védelem* a lehallgatás ellen.
- Tartalom integritása: védelem a megváltoztatás ellen.

Tanúsítvány

Egy megbízható harmadik fél (Certificate Authority) igazolja a szerver hitelességét. Ez nem feltétlen egy szervezet jelent, lehet akár több is, egy láncban, egészen a gyöker tanúsítvány-kiadói (Root CA).

Részei:

- **Signature Algorithm:** aláíráshoz használt algoritmus
- **Issuer:** Kiállító
- **Valid from-to:** érvényességi idők
- **Subject:** Kinek állították ki
- **Public key:** Maga a publikus kulcs
- **Thumbprint:** Ez alapján is kereshető lenyomat
- **SAN:** További FQDN, amikre érvényes lesz a tanúsítvány
 - aldomainek esetén max +1 mélységig

Tanúsítványkészítés folyamata:

1. Kérelem (**Certificate Signing Request**) összeállítása
 - Kulcspár generálása a szerveren, openssl-el
 - A privát részét nem küldi el a szerver a CA-nak, az titkos marad
 - Szervezeti egység adatainak megadása
 - Egyéb adatok (signature algorithm, SAN, stb.)
2. Kérelem feltöltése a CA-hoz, aki ezután meggyőződik róla, hogy megbízható-e a szerver, majd ha rendben találja, visszaküldi az aláírt tanúsítványt a szervernek
 - A CA csak azt igazolja, hogy a nyilvános kulcs az adott tulajdonosé.
3. A visszakapott adatokkal befejezi a szerver az igénylési folyamatot.
4. Hozzáadja a szerver a tanúsítványt az adott weboldalhoz, létrejön a HTTPS binding.

Önaláírt tanúsítványok

- 👍 Gyors, olcsó, tetszőlegesen testreszabható, titkosítja a kapcsolatot
- 👎 Nem azonosítja a szervert; kizárható man-in-the-middle támadással
- 👎 Arra tanítja a felhasználókat, hogy elfogadják a nem hiteles tanúsítványt is.

Mikor érvényes egy tanúsítvány?

Ha:

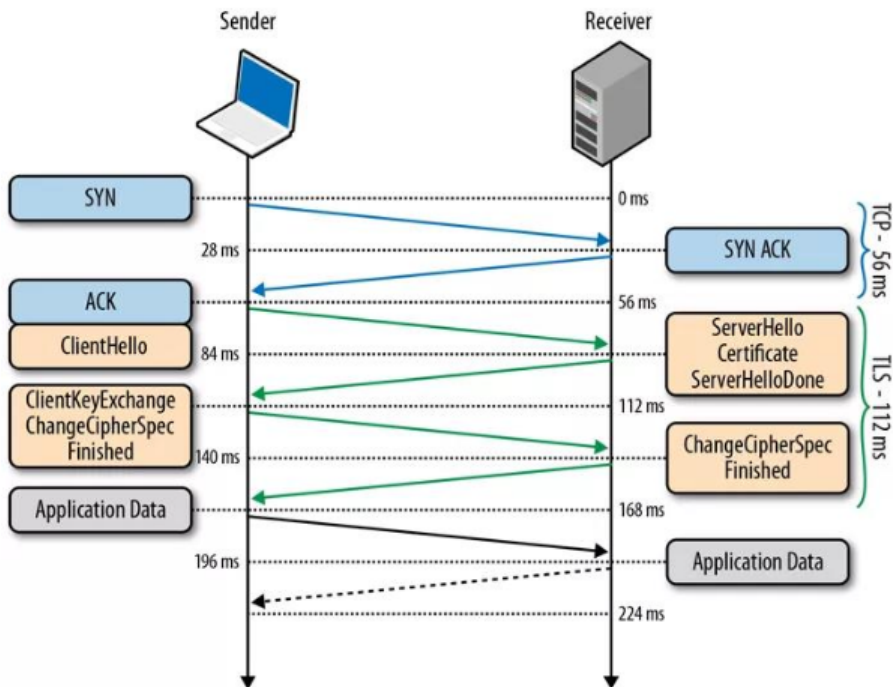
- ✅ Hiteles a kiállító
 - Ehhez a böngészőnek meg kell bíznia a CA lánc minden szereplőjében
- ✅ Nem járt le.
 - Egy tanúsítvány általában 1-3 évig érvényes
- ✅ Az aktuális szerver számára állították ki

- A tanúsítvány Subject mezőjének meg kell egyeznie az oldal betöltéséhez használt FQDN-nel

✓ Nem vonták vissza

- Kompromittálódhat a tanúsítvány, vagy a CA (pl ellopják a CA privát kulcsát, és elkezdnek vele aláírogatni rosszindulatú oldalakat).

Kulcscsere folyamata



Lejjebb látható, hogy egyes protokollokban egyes üzenetek összevonódhatnak.

HTTP Protokollok, TLS

TLS 1.2

- 2 network roundtrip

TLS 1.3

- 1 roundtrip
- Szigorúan korlátozza a használható titkosítási algoritmusokat. A kliens így azonnal kitalálhatja, hogy a szerver melyiket fogja használni

HTTP/3 QUIC

- HTTP + kriptográfiai handshake egyben. Önmagába foglalja a TLS 1.3-at, így nem lehet nélküle használni.

