DNS

Struktúrája:

* **Domain Name Space**: Maga a DNS hierarchia, tartománynevek szervezésének az alapja, a tartománynevek pontokkal elválasztott részletek sorozata melyek az internet egyes szolgáltatásait vagy számítógépeket, szerverek azonosítanak.
* **Domain Name Server**: Névszerverek, ezek tárolják a tartományokat és azokhoz tartozó IP címeket. (lehetővé teszi a név alapján való keresést (IP cím ismerete nélkül))
* **DNS-Resolver**: Szoftver, mely megkeresi egy tartomány név alapján az ahhoz tartozó IP címet. (alkalmazások által küldött kéréseket kezel)
* **Root DNS Server**: Tartalmazzák a teljes tartománynév-területet lefedő DNS-rekordokat és irányítják a DNS-kéréseket. (DNS hierarchia legtetején vna)
* **Top-Level Domain (TLD) Servers**: Ezen szerverek kezelik a TDL-eket (pl.: .com,.hu,.org) minden szerver a saját specifikus tartományát kezeli. (előzőleg felsorolt tartományok)
* **Authoritative Name Servers**: Ezen szerverek felelősek egy adott tartománynévhez rendelt IP-címek vagy egyéb DNS-rekordok tárolásáért és kezeléséért, például az "example.com" tartománynevéhez tartozó IP-címet adják vissza DNS-kérések esetén.

Működése:

1. Amikor egy felhasználó a böngészőjében vagy más alkalmazásban egy tartománynévet (pl. [www.vanenet.](http://www.vanenet.)hu) ír be, a DNS-feloldó a rendszerbe küld egy DNS-kérést.
2. A DNS-feloldó elkezdi a DNS-kérés feldolgozását, kezdve a helyi cache-ek ellenőrzésével. Ha a keresett rekordot megtalálja a cache-ben, akkor azonnal visszaadja az IP-címet, különben folytatja a keresést.
3. Ha a cache nem tartalmazza a keresett rekordot, akkor a DNS-feloldó továbbítja a kérést a gyökér DNS-szerverekhez.
4. A gyökér DNS-szerverek átirányítják a kérést a megfelelő TLD-szerverekhez.
5. A TLD-szerverek továbbítják a kérést az adott tartományt kezelő szerződéses névszerverekhez.
6. A szerződéses névszerverek visszaküldik az IP-címet vagy a kért DNS-rekordot a DNS-feloldónak.
7. A DNS-feloldó visszaadja az IP-címet a böngészőnek vagy alkalmazásnak, és ezzel lehetővé teszi az internetes erőforrás elérését.

REST és SOAP API

|  |  |
| --- | --- |
| SOAP | REST |
| A SOAP web provisioning szolgáltatások megváltoztatása gyakran bonyolult kódváltozással jár a kliens oldalon. | A REST webprovisioningban a szolgáltatások megváltoztatása nem igényel változtatást a kliens oldali kódban. |
| Nagyobb a terhelése | Könnyűsúlyú, kifejezetten könnyű adatátvitelt biztosít egy a legtöbb esetben ismert felületen, - az URI-n keresztül. |
| Bináris csatolmány elemzést igényel. | A REST közvetlenül támogatja az összes adattípust. |
| A SOAP nem igazán barátságos a vezeték nélküli infrastruktúrához. | A REST barátságos a vezeték nélküli infrastruktúrához. |
| A SOAP web szolgáltatások mindig XML adatot adnak vissza. | Míg a REST webes szolgáltatások rugalmasságot biztosítanak a visszaadott adattípusok tekintetében. |
| SOAP válaszként sokkal több sávszélességet használ fel, mivel egy SOAP válasz akár több mint tízszer annyi bájtot is igényelhet, mint a REST. | Kevesebb sávszélességet használ fel, mert a válasz könnyűsúlyú. |
| A SOAP kérések POST módszert használnak, és egy összetett XML kérést igényelnek a létrehozáshoz, ami megnehezíti a válasz gyorsítótárazását. | A RESTful API-k egyszerű GET kérésekkel fogyaszthatók, köztes proxy szerverek / fordított proxy-k könnyen gyorsítótárba helyezhetik válaszaikat. |
| A SOAP HTTP-alapú API-kat használ, amelyeket egy vagy több HTTP URI-ként tesznek elérhetővé, és a tipikus válaszok XML / JSON formátumban érkeznek. A válasz sémák egyediek az adott objektumhoz. | A REST másrészről a standardizált URI-k használatának elemeit is hozzáadja, és fontosnak tartja az alkalmazott HTTP műveletet (pl. GET / POST / PUT stb.). |
| Nyelv-, platform- és szállításfüggetlen. | Nyelv- és platformfüggetlen. |
| A distributed computing környezetek kezelésére tervezve. | Egy ponttól-pontig kommunikációs modellt feltételez - nem alkalmas elosztott számítási környezetre, ahol az üzenet áthaladhat egy vagy több köztes csatornán. |
| Nehéz fejleszteni, eszközök szükségesek hozzá. | A webes szolgáltatások fejlesztése sokkal egyszerűbb, mint a SOAP esetében. |
| A webes szolgáltatások terén uralkodó szabvány, és ezért jobb támogatást kap más szabványoktól (WSDL, WS) és eszközöktől a szolgáltatóktól. | A biztonság, a politika, az üzenetküldés megbízhatósága stb. terén hiányoznak a szabványos támogatások, így azok a szolgáltatások, amelyeknek kifinomultabb követelményeik vannak, nehezebben fejleszthetők. |

Szoftverfejlesztéshez használatos hálózati protokollok:

**SSH (Secure SHell):**

Az SSH (Secure Shell) egy kriptografikus hálózati protokoll, mely biztosítja a biztonságos kommunikációt két távoli számítógép között. A főbb tulajdonságai közé tartozik a titkosított adatkommunikáció, széleskörű hitelesítési lehetőségek biztosítása (mint például jelszóalapú vagy kulcs-alapú hitelesítés), távoli hozzáférés lehetősége. Az SSH segítségével lehetővé válik a szerverek távoli kezelése, fájlok biztonságos átvitele és tárolása, például a man-in-the-middle támadásokkal szemben. Az SSH a TCP, UDP, STCP portokat használhatja.

**FTP (File Transfe):**

Az FTP két módban működhet: aktív és passzív.

* Az aktív módban a kliens hallgat a szerver felől érkező adatkapcsolatokra egy M porton. A szerver kezdeményezi a kapcsolatot az ügyféllel a 20-as portról, azaz az FTP szerver adatportjáról.
* Ha a kliens tűzfal mögött van és nem tud fogadni bejövő TCP kapcsolatokat, passzív módot használhat. Ebben a módban a kliens a vezérlőkapcsolaton keresztül küld egy PASV parancsot a szervernek, majd kap egy szerver IP-címet és portszámot, amelyet aztán az ügyfél használ egy tetszőleges kliensportból a kapott szerver IP-címre és portszámra adatkapcsolat nyitásához.

Az FTP szerver ASCII-ben háromjegyű státuszkódokkal válaszol a vezérlőkapcsolaton, opcionális szöveges üzenettel. Például "200" (vagy "200 OK") azt jelenti, hogy a legutóbbi parancs sikeres volt.

Az FTP-nek két portra van szüksége (egy küldéshez és egy fogadáshoz), mivel eredetileg a Network Control Protocol (NCP) felett működött, amely két portcímet használt kétirányú kommunikációhoz.

Az FTP adatokat általában úgy továbbítja, hogy a szerver visszakapcsolódik az ügyfélhez, miután az ügyfél elküldte a PORT parancsot. Ez problémás a NAT és a tűzfalak számára, amelyek nem engedik az internet felől érkező kapcsolatokat a belső hosztok felé. Két megoldás létezik a probléma megoldására: az FTP kliens és szerver használhatják a PASV parancsot vagy a NAT módosíthatja a PORT parancs értékeit egy alkalmazásszintű átjáró segítségével.

**SMTP (Simple Mail Transfer Protocol):**

Az e-mailt egy levelezőkliens (mail user agent, MUA) küldi el egy levelezőszerverre (mail submission agent, MSA) SMTP protokollon keresztül a TCP 587-es portján. Az MSA továbbítja az e-mailt a levelezőküldő ügynökhöz (mail transfer agent, MTA). A levél feldolgozása lehet egy gépen történő egyszerű vagy több gépre szétosztott feladat; ha több gépen történik a feldolgozás, akkor a levelezőügynökök SMTP-t használnak az üzenetek továbbítására egymás között.

A határ MTA DNS-t használ a címzett domain MX (mail exchanger) rekordjának kereséséhez, majd az alapján választ címzett szervereket és kapcsolódik hozzájuk az e-mail cseréhez. Az üzenetátvitel egyetlen kapcsolatban történhet két MTA között, vagy több köztes rendszeren keresztül. A fogadó SMTP szerver az utolsó célállomás lehet, egy köztes "relé" (amely tárolja és továbbítja az üzenetet), vagy egy "gateway" (amely más protokollt használ az üzenet továbbítására, mint az SMTP).

Az SMTP az üzenet átvitelét, nem pedig annak tartalmát definiálja. Az SMTP működése kapcsolatorientált, szöveges protokoll, amelyben a feladó és a fogadó SMTP szerver parancsokat vált, és adatokat szolgáltat egy megbízható, rendezett adatfolyam csatornán keresztül, általában egy TCP kapcsolaton. Az SMTP művelet egy háromparancs/parancs-válasz szekvenciából áll: MAIL parancs, RCPT parancs, DATA parancs.

Az e-mail kiszolgáló a helyi kézbesítéshez egy levelezési szállítási ügynöknek (mail delivery agent, MDA) adja át az üzenetet. Az IMAP és a POP az e-mail üzenetek elérését és kezelését szolgálja, míg az SMTP az üzenetek továbbítását biztosítja.

**HTTP és HTTPS:**

HTTP a kliens–szerver modellben működő kérés–válasz protokollként funkcionál. Például egy webböngésző lehet a kliens, míg egy számítógépen futó, egy vagy több webhelyet hostoló folyamat, az úgynevezett webszerver, lehet a szerver. A kliens egy HTTP kérésüzenetet küld a szervernek. A szerver, amely erőforrásokat szolgáltat, például HTML fájlokat és egyéb tartalmat, vagy más funkciókat végez a kliens nevében, válaszüzenetet küld vissza a kliensnek. A válasz tartalmazza a kérés teljesítésére vonatkozó információkat, valamint a kért tartalmat az üzenettestben.

A webböngésző egy felhasználói ügynök (UA) példája. A keresőszolgáltatók indexelő szoftverei (web crawler-ek), hangalapú böngészők, mobilalkalmazások és más szoftverek is példák lehetnek egy felhasználói ügynökre, amelyek hozzáférnek, feldolgozzák vagy megjelenítik a webes tartalmat.

Az HTTP lehetővé teszi az intermediális hálózati elemek számára, hogy javítsák vagy lehetővé tegyék a kommunikációt a kliensek és a szerverek között. A nagy forgalmú weboldalak gyakran előnyt élveznek a webgyorsítótár-szerverektől, amelyek a felső szintű szerverek nevében szolgáltatják a tartalmat a válaszidő javítása érdekében. A webböngészők korábban letöltött webes erőforrásokat gyorsítótároznak és újrahasználják őket a hálózati forgalom csökkentése érdekében. Az HTTP proxy szerverek a privát hálózatok határainál megkönnyíthetik a kommunikációt olyan kliensek számára, amelyeknek nincs globálisan routelhető címük, külső szerverekkel való üzenetek továbbításával.

Az HTTP fejlécek egy részét (amelyeket HTTP kérések/válaszokban találunk) úgy kezelik, hogy hop-by-hop, míg más HTTP fejléceket végponttól-végpontig kezelnek (csak a forrás kliens és a célwebszerver által kezelt fejléceket).

Az HTTP egy alkalmazási rétegű protokoll, amelyet az Internet protokollhalmazának keretein belül terveztek. A legújabb verzió, az HTTP/3 esetében a Transmission Control Protocol (TCP) már nem használatos, de a régebbi verziók még mindig nagyban használják, és általában TCP-t alkalmaznak. Módosították azonban úgy, hogy nem megbízható protokollokat is használjanak, mint például az User Datagram Protocol (UDP), amelyet az HTTP/3 is (közvetetten) használ, például az HTTPU és a Simple Service Discovery Protocol (SSDP) esetében.

Az HTTP erőforrásokat Uniform Resource Locators (URLs) alapján azonosítanak és helyeznek el a hálózaton, a Uniform Resource Identifiers (URI-k) séma http és https felhasználásával. Az URI-ket, ahogy az RFC 3986-ban meghatározták, hiperhivatkozásként kódolják a HTML dokumentumokban, hogy interaktív hipertext dokumentumokat képezzenek.

Az HTTPS azonos szintaktikát használ, mint az HTTP, de kiegészíti azt az SSL/TLS titkosítási réteggel a forgalom védelméhez. A SSL/TLS különösen alkalmas az HTTP-hez, mivel védelmet nyújthat, még akkor is, ha csak az egyik kommunikációi fél azonosított. Az HTTPS biztonságos csatornát teremt egy bizonytalan hálózaton át. Böngészők alapvetően megbíznak az HTTPS webhelyekben előtelepített tanúsítványok által. Az HTTPS különösen fontos az bizonytalan és manipulált hálózatok esetén, mint például a nyilvános Wi-Fi hozzáférések, valamint a Tor hálózaton keresztüli kapcsolatoknál. Az HTTPS használata szintén lehetővé teszi az új HTTP verziók, mint az HTTP/2 és HTTP/3, valamint azok elődeinek, a SPDY és QUIC használatát, amelyek a lapbetöltési időt, méretet és késleltetést csökkentik. Az HTTPS használatához ajánlott az HTTP Strict Transport Security (HSTS) alkalmazása, hogy megvédje a felhasználókat a közbeékeléses támadásoktól.

Végpontok Könyvek keresése URI: /books Módszer: GET Paraméterek: title: A könyv címe (opcionális) author: A szerző neve (opcionális) genre: A könyv műfaja (opcionális) Válasz: A könyvek listája, amelyek megfelelnek a keresési feltételeknek. Könyv részletei URI: /books/{bookId} Módszer: GET Válasz: A könyv részletei, beleértve a címét, szerzőjét, műfaját, és elérhetőségét. Könyv kölcsönzése URI: /books/{bookId}/borrow Módszer: POST Válasz: Kölcsönzési státusz és a kölcsönzés lejárati ideje. Könyv visszaadása URI: /books/{bookId}/return Módszer: POST Válasz: Visszaadási státusz. Hiba kezelés 404 Not Found: Ha a keresett erőforrás nem található. 400 Bad Request: Ha a kérés formátuma helytelen. 401 Unauthorized: Ha a hitelesítési token érvénytelen vagy lejárt. 500 Internal Server Error: Szerver oldali hiba esetén. 2. Adatmodell Könyv id: Egyedi azonosító (UUID) title: A könyv címe author: A szerző neve genre: A könyv műfaja status: Kölcsönzött vagy elérhető 3. Technológiai stack Backend: Node.js Express keretrendszerrel Adatbázis: MongoDB Hitelesítés: OAuth 2.0 szerver 4. Fejlesztési és üzemeltetési megfontolások Az API-nak skálázhatónak kell lennie, és képesnek kell lennie kezelni a nagy felhasználói forgalmat. Biztonsági intézkedések, mint például HTTPS, adatvédelem, és az API sebezhetőségeinek rendszeres felülvizsgálata. Monitorozás és naplózás a rendszer állapotának és teljesítményének nyomon követésére. Ez az API terv egy alap, amelyet további igényekhez mérten lehet bővíteni és finomítani.

**Http hibakódok**

* (1xx – tájékoztatás)
* 2xx – sikeres kérés
* 3xx – átirányítás
* 4xx – klienshiba
* 5xx – szerverhiba

**2xx**

* 200 – OK – sikeres kérés
* 201 – Created – sikeres kérés, tartalom létrehozásra került
* 202 – Accepted – kérés elfogadva, de a feldolgozás nem fejeződött be
* 204 – No Content – kérés sikeresen feldolgozva, nem küld vissza a szerver tartalmat

**3xx**

* 300 - Multiple Choices –több opció felkínálása, amiből a kliens választhat
* 301 – Moved Permanently – véglegesen megváltozott a tartalom URL-je
* 302 – Found – átmenetileg megváltozott a tartalom URL-je

**4xx**

* 400 – Bad Request – kérés nem feldolgozható kliensoldali hiba miatt, pl. szintaktikai hiba, túl nagy méret
* 401 – Unauthorized – a szerver értelmezte a kérést, de megtagadta a hozzáférést, mert hitelesítés szükséges
* 403 – Forbidden – a szerver értelmezte a kérést, és megtagadta a hozzáférést
* 404 – Not Found – a tartalom nem található
* 405 – Method Not Allowed – nem támogatott http metódussal érkezett a kérés

**5xx**

* 500 – Internal Server Error – váratlan hiba lépett fel, ezért nem teljesíthető a kérés
* 501 – Not Implemented – a szerver nem ismerte fel a http metódust, vagy pedig nem képes teljesíteni a kérést
* 502 – Bad Gateway – a szerver átjáróként vagy proxy-ként működött, és érvénytelen választ kapott az upstream szervertől
* 503 – Service Unavailable – a szerver nem tud kéréseket fogadni, pl. túlterhelés, karbantartás miatt
* 504 – Gateway Timeout - a szerver átjáróként vagy proxy-ként működött, és nem kapott időn belül választ az upstream szervertől
* 505 – Http Version Not Supported

**Http metódusok**

Leggyakoribb metódusok a GET és POST, ezen kívül még létezik PUT, HEAD, DELETE, PATCH, OPTIONS, CONNECT, TRACE

* GET: Reprezentációt kér a meghatározott tartalomból. Csak adatok kérésére használják. Cache-elhető, a böngésző előzmények között maradnak, hosszuk korlátozott.
* POST: Adatot küld a szervernek, hogy tartalom kerüljön létrehozásra, frissítésre. Nem cache-elhető, nem marad az előzmények között és nincs korlátozás a hosszukra.
* PUT: A cél összes reprezentációját lecseréli a tartalmára (felülír). Adatot küld a szervernek a POST-hoz hasonlóan. A különbség az, hogy ugyanannak a PUT requestnek többszöri meghívására egyszer lesz a tartalom létrehozva. A POST ezzel szemben többször is létrehozza.
* HEAD: A GET-tel majdnem megegyezik a funkciója azzal a különbséggel, hogy a válasz nem tartalmaz response body-t.
* DELETE: Törli a megadott tartalmat.
* OPTIONS: A szerver által támogatott metódusok listáját adja vissza.
* PATCH: Részleges változtatásokat végez a tartalmon.
* CONNECT: Tunnelt hoz létre a szerverrel.
* TRACE: Visszaküldi a kapott kérést (loopback) a tartalom elérési útjával.

**HTTP fejlécek szerepe**

A http fejlécek lehetővé teszik, hogy további információkat adjon át a szerver és a kliens egymásnak egy http kéréssel vagy válasszal. Csoportosításuk: request, response, representation, payload.

* User-Agent: Request fejléc, ami felismeri a kérést küldő user agentről\* a következő információkat: alkalmazás neve, verziószám, operációs rendszer, egyéb információk  
  pl. User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36   
  \*user agent: egy személy reprezentálására használt program
* Authorization: Request típusú, általában akkor használják, amikor a user agent\* megpróbál elérni hitelesítést igénylő védett tartalmat. Ez a fejléc tartalmazza az autentikációhoz szükséges hitelesítő adatokat (felhasználónév, jelszó).  
  pl. Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
* Set-Cookie: Http-response fejléc, amit a szerver használ egy cookie elküldésére a user agenthez. Tartalmazza a nevet, értékét, és egyéb információt pl. lejárati dátum. A cookie-t a válasz fogadása után lokálisan tárolják, és automatikusan csatolva lesz a további kérésekhez ugyanarra a domainre.  
  pl. Set-Cookie: sessionId=abc123; Expires=Wed, 09 Jun 2024 10:18:14 GMT; Path=/
* Accept: Request típusú. A kliens használja annak jelzésére a szerver felé, hogy milyen formátumú és típusú tartalmat fogad el a szervertől. Opcionálisan tartalmazhat paraméterként karakterkészletet, nyelvet stb.  
  pl. Accept: text/html
* Content-Type: A szerver használja arra, hogy jelezze a küldött média típusát a kliens felé. Használják pl. PUT, POST requesteknél.  
  pl. Content-Type: application/x-www-form-urlencoded
* Content-Disposition: Response header, ami jelzi, hogy a tartalmat inline (weboldalként vagy weboldal részeként) kell megjeleníteni vagy pedig csatolni, és letölteni és menteni lokálisan.  
  pl. Content-Disposition: attachment; filename="example.txt",  
  Content-Disposition: inline
* Location: 3xx kódú válaszokban, átirányításkor használják, vagy pedig 201-nél, létrehozásnál.   
  pl. Location: https://example.com/new-page

Példa: Könyvtárnyilvántartó programhoz API készíteni.

1. Adatmodell kialakítása: Könyv:
   1. ID: Egyei azonosító
   2. author: szerző
   3. title: cím
   4. genre: műfajok
   5. status: kivan-e kölcsönözve
2. Felhasznált technológiák:
   1. Backend: Java
   2. Adatbázis: PostgreSQL
3. API specifikáció
   1. Könyv adatai:
      * Könyvek keresése
      * URI: /books/{ID}/data
      * Módszer: GET
      * Paraméterek:
        1. title
        2. author
        3. genre
   2. Könyv hozzáadása:
      * URI: /books/add
      * Paraméter: ID
      * Módszer: POST
   3. Könyv törlése
      * URI: /books/{ID}/delete
      * Módszer: DELETE
      * Válasz: Törlési státusz.
   4. Könyv frissítése
      * URI: /books/{ID}/update
      * Módszer: PUT
   5. Hiba kezelés
      * 404 Not Found: Ha a keresett erőforrás nem található.
      * 400 Bad Request: Ha a kérés formátuma helytelen.
      * 401 Unauthorized: Ha a hitelesítési token érvénytelen vagy lejárt.
      * 500 Internal Server Error: Szerver oldali hiba esetén.

(Forrás: https://developer.mozilla.org/)