Engineer

HTTP

⇄ forespørgslen og svaret som webben bygger på

Hvad det er

HTTP, Hypertext Transfer Protocol, er de regler, en browser, en app eller et script følger, når den beder en server om noget og får et svar tilbage. Kører samtalen inde i en krypteret TLS-forbindelse, hedder den samme protokol HTTPS. En udveksling består altid af to dele. Klienten sender en forespørgsel med en metode som `GET` eller `POST`, det mål, den vil have fat i, en række headerfelter og nogle gange en krop. Serveren svarer med en statuskode på tre cifre, sine egne headerfelter og som regel en krop. Når en side indlæses, et billede hentes eller ét program kalder et andet, er det den samme udveksling, der gentages.

```
GET /pricing/ HTTP/1.1
Host: example.com
Accept: text/html

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=600
```

Specifikationen kalder protokollen tilstandsløs. Hver forespørgsel skal indeholde alt, hvad serveren behøver for at forstå den, og protokollen husker intet fra den ene udveksling til den næste. Login, indkøbskurve og sessioner bygges ovenpå, typisk med cookies: et headerfelt, serveren sætter én gang, og som klienten sender med på alle senere forespørgsler.

Ordforrådet er lille, men præcist. `GET` og `HEAD` er sikre metoder; de beder om en repræsentation og forventes ikke at ændre noget. `PUT` og `DELETE` er idempotente, så den samme forespørgsel sendt to gange efterlader serveren i samme tilstand som én gang. `POST` lover ingen af delene. Statuskoderne er

grupperet efter første ciffer: 2xx betyder succes, 3xx omdirigering, 4xx en fejl i forespørgslen og 5xx en fejl på serveren. En klient, der møder en kode, den ikke kender, ved derfor alligevel nogenlunde, hvad der skete.

Betydningen har ligget fast, mens overførselsformatet er skiftet ud to gange. HTTP/1.1 skriver beskederne som tekstlinjer over en TCP-forbindelse, og hver forbindelse leverer sine svar ét ad gangen og i rækkefølge. HTTP/2 beholder metoder og felter, men pakker dem i binære rammer og fletter mange forespørgsler sammen på én forbindelse. HTTP/3 viderefører den samme semantik over QUIC, en transport bygget på UDP, som undgår TCP's blokering forrest i køen på transportlaget mellem uafhængige strømme. Siden 2022 har IETF udgivet semantikken i ét dokument og hver versions overførselsformat i et andet, så »hvad betyder 404« og »hvordan er byterækkefølgen opbygget« nu har hvert sit svar.

HTTP eller HTTPS er altså ikke et valg mellem to protokoller. HTTPS er HTTP med TLS nedenunder, som krypterer udvekslingen og beviser serverens navn. WebSocket begynder som en almindelig HTTP-forespørgsel og opgraderer derefter forbindelsen til en kanal, hvor begge parter kan sende, og gRPC er en konvention for fjernkald, der kører oven på HTTP/2.

Brug begrebet i en beslutning

Brug »Hvad er HTTP?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Wi-Fi, firewalls og fjernadgang sat op, så medarbejdere kan arbejde hvor som helst, og kun de rette kommer ind.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har administreret 30 Ubuntu Linux VPS-instanser — 30 Linux-servere driftet, med en plan for gendannelse

Har administreret 30 Ubuntu Linux VPS-instanser Kontakt os →

Hvorfor det betyder noget

Stort set alle systemer, der taler med andre over internettet, bruger allerede HTTP. Det interessante valg er derfor, om man bruger det, protokollen definerer, eller behandler den som et rør og bygger resten selv.

Der er god grund til at bruge den fuldt ud, for en lang kæde af software forstår allerede ordforrådet. Et klientbibliotek må gerne prøve en `GET` igen, når den løber tør for tid, men skal lade en `POST` være, og det samme gælder en proxy undervejs. En cache må gemme et svar mærket `max-age=600` i ti minutter og give

det videre til den næste læser uden at spørge serveren. Overvågning tæller 5xx-svar som fejl uden at få det at vide. Et API, der sender hvert kald som en `POST` til én adresse og melder fejl inde i et 200-svar, giver afkald på alt det, og hver klient må lære netop det API's private regler.

Modellen har sine grænser. Forespørgsel og svar passer til en klient, der ved, hvad den vil have, og langt dårligere til en server, der har nyt at fortælle. Derfor går live-opdateringer som regel over WebSocket, server-sent events eller gentagne forespørgsler.

Ukrypteret HTTP på det åbne internet er et kortere spørgsmål. Browsere markerer det som usikkert, alle på vejen kan læse og ændre indholdet, og certifikater er gratis, så offentlig trafik hører hjemme på HTTPS. Valget mellem HTTP/1.1, HTTP/2 og HTTP/3 er mest en indstilling på serveren eller kanten foran den. Applikationens kode forbliver den samme, fordi hver forespørgsel betyder det samme i alle tre.

Sådan ser det ud i praksis

Det tydeligste eksempel her på sitet er et API, der blev flyttet over på et framework, som er bygget op omkring HTTP's egen form. HTTP-API'et flyttede fra Fiber til Huma v2, der kører oven på Fiber. Hvert endpoint erklæres med structs for input og output, og valideringen kommer fra typerne i stedet for fra håndskrevne tjek. Ud af de samme erklæringer kommer en OpenAPI-beskrivelse, og et tjek i CI sammenligner de ruter, serveren registrerer, med dem, beskrivelsen annoncerer. Ved 649 paths og 760 operationer er pariteten 100 %, og den holdes, fordi en rute uden beskrivelse får buildet til at fejle. Præcis to endpoints blev på rå Fiber, nemlig WebSocket-endpointene, hvor Humas model med forespørgsel og svar ikke passer.

Sitet bruger selv indholdsforhandling. Hver side udgives både som HTML og som Markdown, og én måde at få Markdown-udgaven på er at sende `Accept: text/markdown` og lade serveren vælge. Samme side forklarer, hvorfor onion-adressen med vilje er almindelig `http://` : adressen indeholder tjenestens offentlige nøgle i kodet form, så kredsløbet er godkendt og krypteret, før der overhovedet bliver talt HTTP.

HTTP dukker også op som værktøj i infrastrukturkode. Da en ekstra server blev sat op, blev begge udbyderes firewalls skrevet direkte mod deres REST-API'er gennem en generisk HTTP-opgave i stedet for en leverandørsamling. Det

fjernede en afhængighed og dens versionsdrift mod at skrive hver forespørgsels form i hånden. Siden nævner også en anden pris: den generiske HTTP-opgave erklærer ingen understøttelse af check-tilstand, så en prøvekørsel af det play, der opretter maskinen, springer hver opgave i det over. Prøven køres derfor mod firewall-playet.

Hvor det går galt

De fleste HTTP-fejl opstår, når et ord i ordforrådet bruges til at betyde noget andet end det, resten af kæden hører.

En fejl, der ligner en succes

Et API svarer 200 og lægger `"error": "card declined"` i kroppen. En cache kan gemme svaret, dashboards tæller det som en succes, og klientens logik for nye forsøg går aldrig i gang. Send den statuskode, der passer, 4xx eller 5xx, og beskriv problemet i kroppen. RFC 9457 definerer en JSON-form til netop det, leveret som `application/problem+json`.

Et link, der sletter

En `GET` ændrer tilstand, for eksempel et afmeldings- eller slettelink i en mail. Crawlere, linkforhåndsvisninger i chatapps, sikkerhedsscannere og browserens forhåndsindlæsning følger alle links på egen hånd, og hver af dem kan udløse handlingen. Det, der ændrer data, hører hjemme bag `POST`, `PUT` eller `DELETE`, eventuelt med en bekræftelsesside, hvis linket skal findes.

En betaling, der trækkes to gange

Klienten sender en `POST`, forbindelsen ryger, før svaret kommer, og klienten prøver igen. Serveren havde allerede behandlet den første. HTTP giver klienten ingen måde at skelne de to tilfælde på, så API'et må gøre det: tag imod en idempotensnøgle i et headerfelt og returnér det gemte resultat, når nøglen kommer igen, eller lad klienten vælge den nye ressources id og oprette den med `PUT`.

En formular, der bliver til en GET

Serveren svarer på en `POST` med 301 eller 302 for at sende den videre. Af historiske grunde tillader specifikationen, at brugeragenten skifter metoden til `GET` ved netop de to koder, og mange gør det, så kroppen forsvinder undervejs. Skal metoden overleve omdirigeringen, er 307 og 308 de rigtige koder.

Den forkerte udgave fra en cache

Én URL leverer HTML eller Markdown, dansk eller engelsk, afhængigt af forespørgslens headerfelter. En cache, der intet har fået at vide, gemmer den udgave, den så først, og giver den videre til den næste læser. Svaret skal nævne de felter, det afhænger af, for eksempel `Vary: Accept`, så cachen gemmer én kopi pr. udgave frem for én pr. URL.

Læs mere

- RFC 9110: HTTP Semantics — metoder, statuskoder og headerfelter, defineret én gang for alle versioner.
- RFC 9111: HTTP Caching — hvornår et gemt svar må genbruges, og hvad `Cache-Control` fortæller en cache.
- RFC 9112: HTTP/1.1 — tekstformatet på ledningen, forbindelser og hvordan en besked afgrænses.
- RFC 9113: HTTP/2 — binære rammer og mange strømme over én forbindelse.
- RFC 9114: HTTP/3 — den samme semantik lagt oven på QUIC-strømme.
- RFC 9457: Problem Details for HTTP APIs — en fast JSON-krop til at beskrive, hvad der gik galt.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er HTTP?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/http/