Engineer

REST

🏛 webbens arkitekturstil, bygget op om ressourcer

Hvad det er

REST, en forkortelse for Representational State Transfer, er den arkitekturstil, Roy Fielding beskrev i kapitel 5 af sin ph.d.-afhandling fra 2000. Det er hverken en protokol eller en standard. Det er et sæt begrænsninger for, hvordan en klient, en server og mellemleddene imellem dem må interagere, og Fielding gør rede for, hvad hver begrænsning giver, og for de flestes vedkommende også, hvad den koster. RESTful er tillægsordet: strengt taget betegner det et system, der overholder alle begrænsningerne. I daglig tale strækkes begge ord langt videre, til næsten ethvert HTTP-API, der udveksler JSON, hvad enten det overholder begrænsningerne nedenfor eller ej.

Det bærende begreb er ressourcen. Fielding definerer den bredt: al information, der kan gives et navn, kan være en ressource, for eksempel et dokument, et billede, en samling af andre ressourcer, en person, ja, endda dagens vejr i en bestemt by. Det, der krydser netværket, er aldrig selve ressourcen, men en repræsentation, en række byte plus metadata, der beskriver de byte, for eksempel et HTML-dokument eller et JPEG-billede, og en klient handler på en ressource ved at sende eller modtage en repræsentation af dens nuværende eller tilsigtede tilstand.

Fielding bygger stilen op fra et tomt sæt begrænsninger og tilføjer dem én ad gangen. Klient-server adskiller brugergrænsefladen fra datalagringen, så de to kan udvikle sig hver for sig. Tilstandsløshed betyder, at hver forespørgsel indeholder alt, hvad der skal til for at forstå den, og ikke trækker på nogen kontekst gemt på serveren, så sessionstilstanden ligger helt hos klienten. Cache kræver, at hvert svar er mærket, eksplicit eller implicit, som cachebart eller ej, så en klient kan genbruge et cachebart svar til en senere, tilsvarende forespørgsel. Ensartet grænseflade, som er beskrevet nedenfor, giver alle komponenter den samme generelle grænseflade. Lagdelt system betyder, at hver komponent kun ser det lag, den taler direkte med, og det giver plads til proxyer, gateways og

firewalls imellem, uden at grænsefladerne på nogen af siderne skal ændres. Kode efter behov lader en server udvide en klient ved at sende den scripts, den kan køre, og Fielding gør den til den eneste valgfri begrænsning, fordi den mindsker synligheden.

Han kalder den ensartede grænseflade det centrale træk, der adskiller REST fra andre netværksbaserede stilarter, og deler den op i fire begrænsninger: ressourcer identificeres, på webben med URL'er; ressourcer manipuleres gennem repræsentationer; beskeder er selvbeskrivende, så deres betydning er synlig for ethvert mellemled; og hypermedier er motoren i applikationstilstanden, hvilket betyder, at klienten bevæger sig fra én tilstand til den næste ved at vælge blandt de overgange, der tilbydes i de repræsentationer, den allerede har, sådan som et menneske følger links i en browser.

Brug begrebet i en beslutning

Brug »Hvad er REST?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Den del, brugerne aldrig ser: hvor data ligger, hvilke regler der gælder, og hvordan andre systemer må tale med den.

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 designet og drevet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tre databasemotorer

Har designet og drevet 6 ETL/ELT-pipelines Kontakt os →

Hvorfor det betyder noget

Hver begrænsning giver noget konkret til gengæld. Tilstandsløshed lader et overvågningssystem forstå en forespørgsel uden at se ud over den, gør det lettere at komme sig efter delvise fejl og lader en server frigive ressourcer, så snart den har svaret. En cache kan fjerne nogle interaktioner helt. Lag lader mellemled fordele belastningen over flere netværk og maskiner og lader én cache i kanten af en organisation betjene alle inden for den. Og fordi den ensartede grænseflade afkobler implementeringerne fra de tjenester, de leverer, kan klient og server hver især ændre sig, uden at den anden skal følge med i samme takt.

Fielding lægger lige så lidt skjul på omkostningerne. Tilstandsløshed gentager data i hver forespørgsel, som en fælles kontekst på serveren ville have sparet. En cache kan udlevere forældede data. Hvert lag lægger overhead og latenstid til. Den ensartede grænseflade forringer effektiviteten, fordi information sendes i en standardiseret form frem for en, der er tilpasset en bestemt applikations

behov. Fielding designede den til overførsel af hypermediedata i store bidder, det almindelige tilfælde på webben, og siger rent ud, at den ikke er optimal til andre former for interaktion. Det er den grund, alternativerne står på. En GraphQL-server tager imod alle forespørgsler på ét endpoint, identificerer sine entiteter med noget andet end URL'er og lader klienten vælge præcis, hvilke felter der kommer tilbage, mens gRPC kalder metoder, der er erklæret i en tjenestedefinition. Hver af dem opgiver en del af den ensartede grænseflade for at passe til en anden slags trafik.

For et HTTP-API følger meget af den ensartede grænseflade med fra starten. HTTP definerer allerede metoder med en aftalt betydning, statuskoder og headerfelter til caching og til betingede forespørgsler. Fordi den betydning er fælles, kan en proxy eller cache undervejs handle på en besked uden at vide noget om det pågældende API, og et API, der bruger delene, som de er defineret, får det maskineri forærende. Et API, der giver sine klienter URL-skabeloner at udfylde i stedet for links at følge, opgiver hypermedie-begrænsningen, og dets klienter bliver bundet til serverens URL-struktur: når strukturen ændres, går de i stykker, og det er netop den kobling, den ensartede grænseflade findes for at fjerne.

Sådan ser det ud i praksis

Udvekslingen nedenfor opretter en ordre og får som svar adressen på den nye ressource i `Location` og et entity tag i `ETag`. RFC 9110 bruger selv et `ETag` i et 201-svar som eksempel: mærket kan bruges som validator i senere betingede forespørgsler, så to klienter ikke i det stille kan overskrive hinandens ændringer.

```
POST /orders HTTP/1.1
Host: api.example.com
Content-Type: application/json

{"item": "notebook", "quantity": 2}

HTTP/1.1 201 Created
Location: /orders/7421
ETag: "v1"
Content-Type: application/json

{"id": 7421, "item": "notebook", "quantity": 2, "status": "open"}
```

Hver metode har en betydning, som ethvert mellemled kan regne med. RFC 9110 definerer GET, HEAD, OPTIONS og TRACE som sikre: klienten hverken beder om eller forventer nogen ændring af tilstanden på serveren. PUT og DELETE er sammen med de sikre metoder idempotente, så flere ens forespørgsler har den samme tilsigtede virkning som én. POST er ingen af delene, og RFC 5789 siger det samme om PATCH. Det er de to egenskaber, der lader en crawler følge links uden at gøre skade, lader en cache hente på forhånd og lader en klient sende en PUT igen, når forbindelsen røg, før svaret nåede frem.

Et REST-API kan styres med intet andet end en generisk HTTP-klient, og her på sitet ses både, hvad det sparer, og hvad det koster. Da en ekstra server blev sat op fra kode, blev begge udbyderes firewalls konfigureret direkte gennem deres REST-API'er med en generisk HTTP-opgave i stedet for gennem en collection fra leverandøren. Det fjerner en afhængighed og de versionsafvigelser, den fører med sig, og prisen er, at hver request skal skrives i hånden. Samme side nævner en anden pris: den generiske HTTP-opgave erklærer ingen understøttelse af check-tilstand, så en prøvekørsel af det play, der opretter maskinen, springer alle opgaver i det over.

Hvor det går galt

En GET ændrer noget. En adresse som `/orders/7421/cancel`, der annullerer en ordre, når den hentes, bryder løftet om, at GET er sikker. RFC 9110 forklarer, hvorfor det løfte findes: så crawlere og caches, der henter på forhånd, kan arbejde uden frygt for at gøre skade. Den første crawler eller forudhentende cache, der støder på linket, annullerer ordren. Læg i stedet handlingen i metoden, som en POST til ordren eller en PATCH af dens status.

Hvert svar er 200 OK. Nogle API'er returnerer 200 for alting og lægger det egentlige udfald, fejl eller ej, i brødteksten. Beskederne holder op med at være selvbeskrivende: caches, proxyer, overvågning og klientbiblioteker læser alle statuskoden, og de ser alle sammen succes. RFC 9457 definerer en standardiseret brødtekst til fejl, `application/problem+json`, med medlemmer som `type`, `title`, `status` og `detail`. Dens `status`-medlem er kun vejledende: serveren skal sende den samme statuskode i selve HTTP-svaret, så generisk HTTP-software, der intet kender til formatet, opfører sig korrekt.

En gentaget POST opretter to ordrer. Når en forbindelse ryger, efter at serveren har behandlet en POST, men før klienten har læst svaret, kan klienten ikke vide, om ordren findes. RFC 9110 siger, at en klient ikke bør gentage en ikke-

idempotent forespørgsel automatisk, medmindre den ved, at forespørgslen alligevel er idempotent, eller kan se, at det første forsøg aldrig blev udført, og at en proxy slet ikke må gøre det. Den bemærker også, at nogle klienter vælger den mere risikable vej og alligevel gentager en POST, når forbindelsen lukkes, før noget af svaret er nået frem, især på en vedvarende forbindelse, der har stået ubrugt hen. En udvej er at lade klienten vælge identifikatoren og sende ordren dertil med PUT, fordi PUT er idempotent: den første forespørgsel opretter ordren med 201, og en gentagelse erstatter den blot med den samme tilstand.

To klienter overskriver hinanden. Begge læser ordren, begge ændrer hvert sit felt, og begge sender det hele tilbage med PUT: den anden skrivning annullerer i det stille den første. RFC 9110 kalder det lost update-problemet. Løsningen er det entity tag, hver klient fik, da den læste ordren, sendt tilbage i `If-Match`. Serveren udfører så kun ændringen, hvis ressourcen stadig er i den tilstand, klienten sidst så; er den ikke det, må serveren ikke udføre den og kan sige det med 412 Precondition Failed. Klienten læser derefter ordren igen, lægger sin ændring ind på ny og prøver en gang til.

```
PUT /orders/7421 HTTP/1.1
Host: api.example.com
If-Match: "v1"
Content-Type: application/json

{"item": "notebook", "quantity": 3, "status": "open"}

HTTP/1.1 412 Precondition Failed
```

Serveren husker samtalen. En server, der gemmer tilstand pr. klient mellem forespørgsler, for eksempel den aktuelle side i en liste eller en halvt udfyldt formular i hukommelsen, har opgivet tilstandsløsheden. En forespørgsel giver ikke længere mening alene, load balanceren skal sende hver klient tilbage til den samme maskine, og en genstart smider tilstanden væk. Fieldings grunde til tilstandsløshed er præcis dem, der går tabt: synlighed, lettere genopretning efter delvise fejl og en server, der kan frigive ressourcer, når den har svaret. Læg tilstanden i forespørgslen, som et sidetoken i URL'en, eller gør den halvfærdige ting til en ressource for sig, for eksempel en ordrekladde, som enhver server kan læse.

Læs mere

- Fielding, chapter 5: Representational State Transfer (REST) — definitionen: hver begrænsning, de egenskaber, den giver, og hvad den ofrer for dem.
- RFC 9110: HTTP Semantics — metoderne, hvilke af dem der er sikre og idempotente, statuskoderne og betingede forespørgsler med `ETag` og `If-Match`.
- RFC 5789: PATCH Method for HTTP — delvis ændring af en ressource, og hvorfor PATCH hverken er sikker eller idempotent.
- RFC 9457: Problem Details for HTTP APIs — en standardiseret JSON-brødtekst til fejlsvar, der bevarer statuskodens betydning.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er REST?« 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/rest/