Engineer

REST

🏛 webbens arkitekturstil, uppbyggd kring resurser

Vad det är

REST, en förkortning av Representational State Transfer, är den arkitekturstil som Roy Fielding beskrev i kapitel 5 av sin doktorsavhandling från 2000. Det är varken ett protokoll eller en standard. Det är en uppsättning begränsningar för hur en klient, en server och mellanleden mellan dem får interagera, och Fielding redogör för vad varje begränsning ger och, för de flesta av dem, vad den kostar. RESTful är adjektivet: strikt sett betecknar det ett system som följer alla begränsningarna. I vardagligt tal tänjs båda orden mycket längre, till nästan vilket HTTP-API som helst som utbyter JSON, vare sig det följer begränsningarna nedan eller inte.

Det bärande begreppet är resursen. Fielding definierar den brett: all information som kan ges ett namn kan vara en resurs, till exempel ett dokument, en bild, en samling andra resurser, en person, ja, till och med dagens väder i en viss stad. Det som korsar nätverket är aldrig själva resursen utan en representation, en sekvens av byte plus metadata som beskriver dessa byte, till exempel ett HTML-dokument eller en JPEG-bild, och en klient agerar på en resurs genom att skicka eller ta emot en representation av dess nuvarande eller avsedda tillstånd.

Fielding bygger upp stilen från en tom mängd begränsningar och lägger till dem en i taget. Klient-server skiljer användargränssnittet från datalagringen, så att de två kan utvecklas var för sig. Tillståndslöshet innebär att varje förfrågan innehåller allt som behövs för att förstå den och inte lutar sig mot något sammanhang som sparats på servern, så att sessionstillståndet ligger helt hos klienten. Cache kräver att varje svar är märkt, uttryckligen eller underförstått, som cachebart eller inte, så att en klient kan återanvända ett cachebart svar för en senare, likvärdig förfrågan. Enhetligt gränssnitt, som beskrivs nedan, ger alla komponenter samma generella gränssnitt. Skiktat system innebär att varje komponent bara ser det skikt den talar direkt med, vilket ger utrymme för proxyer, gatewayer och brandväggar emellan utan att gränssnitten på någon av

sidorna behöver ändras. Kod på begäran låter en server utöka en klient genom att skicka skript som den kan köra, och Fielding gör den till den enda valfria begränsningen eftersom den minskar synligheten.

Han kallar det enhetliga gränssnittet det centrala drag som skiljer REST från andra nätverksbaserade stilar och delar upp det i fyra begränsningar: resurser identifieras, på webben med URL:er; resurser hanteras genom representationer; meddelanden är självbeskrivande, så att deras betydelse syns för varje mellanled; och hypermedia är motorn i applikationstillståndet, vilket betyder att klienten går från ett tillstånd till nästa genom att välja bland de övergångar som erbjuds i de representationer den redan har, på samma sätt som en människa följer länkar i en webbläsare.

Använd begreppet i ett beslut

Använd ”Vad är REST?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Den del användarna aldrig ser: där data lagras, vilka reglerna är och hur andra system får kommunicera med den.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Utformade och drev 6 ETL/ELT-pipelines — 6 ETL-pipelines över tre databasmotorer

Utformade och drev 6 ETL/ELT-pipelines Hör av er →

Varför det spelar roll

Varje begränsning ger något konkret tillbaka. Tillståndslöshet låter ett övervakningssystem förstå en förfrågan utan att se längre än till den, gör det lättare att återhämta sig efter partiella fel och låter en server frigöra resurser så snart den har svarat. En cache kan ta bort vissa interaktioner helt. Skikt låter mellanled fördela lasten över nätverk och maskiner och låter en enda cache i utkanten av en organisation betjäna alla innanför. Och eftersom det enhetliga gränssnittet frikopplar implementationerna från de tjänster de levererar kan klient och server ändras var för sig, utan att den andra behöver hänga med i samma takt.

Fielding är lika öppen med kostnaderna. Tillståndslöshet upprepar i varje förfrågan data som ett gemensamt sammanhang på servern hade sparat. En cache kan lämna ut inaktuella data. Varje skikt lägger till overhead och latens. Det enhetliga gränssnittet försämrar effektiviteten, eftersom informationen överförs i en standardiserad form i stället för en som är skräddarsydd efter ett visst programs behov. Fielding utformade det för överföring av hypermediadata i

stora stycken, det vanliga fallet på webben, och säger rakt ut att det inte är optimalt för andra slags interaktion. Det är den grunden alternativen står på. En GraphQL-server tar emot alla förfrågningar på en enda slutpunkt, identifierar sina entiteter med något annat än URL:er och låter klienten välja exakt vilka fält som kommer tillbaka, medan gRPC anropar metoder som deklareras i en tjänstedefinition. Var och en av dem ger upp en del av det enhetliga gränssnittet för att passa en annan sorts trafik.

För ett HTTP-API kommer mycket av det enhetliga gränssnittet på köpet. HTTP definierar redan metoder med överenskommen betydelse, statuskoder och huvudfält för cachning och för villkorliga förfrågningar. Eftersom betydelsen är gemensam kan en proxy eller cache på vägen agera på ett meddelande utan att veta något om det specifika API:et, och ett API som använder delarna så som de är definierade får det maskineriet gratis. Ett API som ger sina klienter URL-mallar att fylla i, snarare än länkar att följa, ger upp hypermediabegränsningen, och klienterna blir bundna till serverns URL-struktur: när strukturen ändras går de sönder, och det är just den kopplingen som det enhetliga gränssnittet finns till för att ta bort.

Så ser det ut i praktiken

Utbytet nedan skapar en order och får som svar den nya resursens adress i `Location` och en entity tag i `ETag`. RFC 9110 använder själv en `ETag` i ett 201-svar som exempel: taggen kan fungera som validator i senare villkorliga förfrågningar, så att två klienter inte i tysthet kan skriva över varandras ändringar.

```
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"}
```

Varje metod har en betydelse som alla mellanled kan lita på. RFC 9110 definierar GET, HEAD, OPTIONS och TRACE som säkra: klienten varken begär eller förväntar sig någon ändring av tillståndet på servern. PUT och DELETE är, liksom de säkra

metoderna, idempotenta, så att flera identiska förfrågningar har samma avsedda effekt som en. POST är ingetdera, och RFC 5789 säger samma sak om PATCH. Det är de två egenskaperna som låter en sökrobot följa länkar utan att ställa till skada, låter en cache hämta i förväg och låter en klient skicka om en PUT vars anslutning bröts innan svaret kom fram.

Ett REST-API kan styras med inget annat än en generisk HTTP-klient, och här på webbplatsen syns både vad det sparar och vad det kostar. När en andra server provisionerades från kod, skrevs båda leverantörernas brandväggar direkt mot deras REST-API:er genom en generisk HTTP-task i stället för genom en collection från leverantören. Det tar bort ett beroende och dess versionsglidning, och priset är att förfrågningarnas form skrivs för hand. Samma sida nämner ett andra pris: den generiska HTTP-tasken deklarerar inget stöd för check mode, så en torrkörning av den play som skapar maskinen hoppar över varje task i den.

Var det går fel

En GET ändrar något. En adress som `/orders/7421/cancel`, som avbeställer en order när den hämtas, bryter löftet om att GET är säker. RFC 9110 förklarar varför det löftet finns: för att sökrobotar och cachar som hämtar i förväg ska kunna arbeta utan att behöva oroa sig för att ställa till skada. Den första sökrobot eller förhämtande cache som stöter på länken avbeställer ordern. Lägg i stället handlingen i metoden, som en POST till ordern eller en PATCH av dess status.

Varje svar är 200 OK. Vissa API:er returnerar 200 för allt och lägger det verkliga utfallet, fel eller inte, i meddelandekroppen. Meddelandena slutar vara självbeskrivande: cachar, proxyer, övervakning och klientbibliotek läser alla statuskoden, och samtliga ser framgång. RFC 9457 definierar en standardiserad kropp för fel, `application/problem+json`, med medlemmar som `type`, `title`, `status` och `detail`. Dess `status`-medlem är bara rådgivande: servern måste skicka samma statuskod i själva HTTP-svaret, så att generisk HTTP-programvara som inte känner till formatet beter sig korrekt.

En upprepad POST skapar två ordrar. När en anslutning bryts efter att servern har behandlat en POST men innan klienten har läst svaret kan klienten inte veta om ordern finns. RFC 9110 säger att en klient inte bör upprepa en icke-idempotent förfrågan automatiskt, såvida den inte vet att förfrågan ändå är idempotent eller kan avgöra att det första försöket aldrig utfördes, och att en proxy inte får göra det alls. Den noterar också att vissa klienter väljer den mer

riskfyllda vägen och ändå upprepar en POST när anslutningen stängs innan något av svaret har kommit fram, särskilt på en beständig anslutning som har stått oanvänd. En utväg är att låta klienten välja identifieraren och skicka ordern dit med PUT, eftersom PUT är idempotent: den första förfrågan skapar ordern med 201, och en upprepning ersätter den bara med samma tillstånd.

Två klienter skriver över varandra. Båda läser ordern, båda ändrar var sitt fält, och båda skickar tillbaka alltihop med PUT: den andra skrivningen gör i tysthet den första ogjord. RFC 9110 kallar det lost update-problemet. Lösningen är den entity tag som varje klient fick när den läste ordern, skickad tillbaka i `If-Match`. Servern utför då ändringen bara om resursen fortfarande är i det tillstånd som klienten senast såg; är den inte det får servern inte utföra den och kan säga det med 412 Precondition Failed. Klienten läser sedan ordern igen, gör om sin ändring och försöker en gång till.

```
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
```

Servern minns samtalet. En server som håller tillstånd per klient mellan förfrågningar, till exempel den aktuella sidan i en lista eller ett halvifyllt formulär i minnet, har gett upp tillståndslösheten. En förfrågan går inte längre att förstå på egen hand, lastbalanseraren måste skicka varje klient tillbaka till samma maskin, och en omstart kastar bort tillståndet. Fieldings skäl för tillståndslöshet är precis de som går förlorade: synlighet, enklare återhämtning efter partiella fel och en server som kan frigöra resurser när den har svarat. Lägg tillståndet i förfrågan, som en sidtoken i URL:en, eller gör det halvfärdiga till en egen resurs, till exempel ett orderutkast, som vilken server som helst kan läsa.

Läs mer

- Fielding, chapter 5: Representational State Transfer (REST) — definitionen: varje begränsning, de egenskaper den ger och vad den offrar för dem.
- RFC 9110: HTTP Semantics — metoderna, vilka av dem som är säkra och idempotenta, statuskoderna och villkorliga förfrågningar med `ETag` och `If-Match`.

- RFC 5789: PATCH Method for HTTP — partiell ändring av en resurs, och varför PATCH varken är säker eller idempotent.
- RFC 9457: Problem Details for HTTP APIs — en standardiserad JSON-kropp för felsvar som behåller statuskodens betydelse.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är REST?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/rest/