Engineer

REST

🏛 Architekturstil des Webs, der auf Ressourcen aufbaut

Was es ist

REST, kurz für Representational State Transfer, ist der Architekturstil, den Roy Fielding im Jahr 2000 in Kapitel 5 seiner Dissertation beschrieb. Es ist weder ein Protokoll noch ein Standard, sondern eine Menge von Bedingungen dafür, wie Clients, Server und die Vermittler dazwischen interagieren dürfen. Fielding nennt den Nutzen jeder Bedingung und bei den meisten auch ihre Kosten. RESTful ist das Adjektiv: Streng genommen beschreibt es ein System, das alle Bedingungen einhält. Im Alltag werden beide Begriffe viel weiter gefasst, bis hin zu fast jeder HTTP-API, die JSON austauscht, unabhängig von den folgenden Bedingungen.

Die zentrale Idee ist die Ressource. Fielding schreibt: „Jede Information, die benannt werden kann, kann eine Ressource sein“: ein Dokument, ein Bild, eine Sammlung weiterer Ressourcen, ein Mensch, sogar das aktuelle Wetter in einer Stadt. Über das Netz geht niemals die Ressource selbst, sondern eine Repräsentation: eine Folge von Bytes samt Metadaten, die diese Bytes beschreiben, etwa ein HTML-Dokument oder ein JPEG-Bild. Ein Client wirkt auf eine Ressource ein, indem er eine Repräsentation ihres aktuellen oder beabsichtigten Zustands sendet oder empfängt.

Fielding baut den Stil aus einer leeren Menge von Bedingungen auf und fügt sie einzeln hinzu. Client-Server trennt die Benutzerschnittstelle von der Datenspeicherung, damit sich beide unabhängig entwickeln können. Zustandslosigkeit bedeutet, dass jede Anfrage alles zu ihrem Verständnis Nötige mitbringt und sich auf keinen gespeicherten Kontext beim Server stützt. Der Sitzungszustand liegt deshalb vollständig beim Client. Die Cache-Bedingung verlangt, jede Antwort ausdrücklich oder implizit als cachefähig oder nicht cachefähig zu kennzeichnen, damit der Client eine cachefähige Antwort für eine spätere gleichwertige Anfrage wiederverwenden kann. Die unten beschriebene einheitliche Schnittstelle gibt jeder Komponente dieselbe allgemeine Schnittstelle. Ein Schichtensystem bedeutet, dass jede Komponente nur die

unmittelbar angesprochene Schicht sieht. Das lässt Raum für Proxys, Gateways und Firewalls dazwischen, ohne die Schnittstellen auf beiden Seiten zu ändern. Code auf Anforderung lässt einen Server einen Client erweitern, indem er Skripte zur Ausführung an den Client sendet. Fielding macht dies zur einzigen optionalen Bedingung, weil es die Sichtbarkeit verringert.

Die einheitliche Schnittstelle nennt er das zentrale Merkmal, das REST von anderen netzbasierten Stilen unterscheidet. Er zerlegt es in vier Bedingungen: Ressourcen werden identifiziert, im Web über URLs; sie werden durch Repräsentationen manipuliert; Nachrichten sind selbstbeschreibend, sodass jeder Vermittler ihre Bedeutung sehen kann; und Hypermedia treibt den Anwendungszustand an. Letzteres heißt, dass der Client seinen nächsten Übergang aus den empfangenen Repräsentationen auswählt, so wie ein Mensch im Browser Links folgt.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist REST?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Jede Bedingung bringt einen bestimmten Vorteil. Zustandslosigkeit lässt ein Überwachungssystem eine Anfrage verstehen, ohne anderswo nachsehen zu müssen. Sie erleichtert die Wiederherstellung nach Teilausfällen und lässt einen Server Ressourcen freigeben, sobald er geantwortet hat. Ein Cache kann manche Interaktionen ganz entfallen lassen. Schichten lassen Vermittler die Last über Netze und Maschinen verteilen und ermöglichen, dass ein Cache am Rand einer Organisation alle im Inneren bedient. Weil die einheitliche Schnittstelle Implementierungen von ihren Diensten entkoppelt, können sich Client und Server jeweils ändern, ohne dass der andere im Gleichschritt folgen muss.

Fielding benennt die Kosten ebenso klar. Zustandslosigkeit wiederholt in jeder Anfrage Daten, die ein gemeinsamer Kontext beim Server eingespart hätte. Ein Cache kann veraltete Daten ausliefern. Jede Schicht erzeugt Zusatzaufwand und Latenz. Die einheitliche Schnittstelle mindert die Effizienz, weil Informationen in

einer standardisierten Form statt zugeschnitten auf eine bestimmte Anwendung übertragen werden. Fielding entwarf sie für grobgranulare Hypermedia-Datenübertragung, den üblichen Fall im Web, und sagt ausdrücklich, dass sie für andere Interaktionsarten nicht optimal ist. Darauf gründen die Alternativen. Ein GraphQL-Server nimmt jede Anfrage an einem einzigen Endpunkt entgegen, identifiziert seine Entitäten anders als durch URLs und lässt den Client genau die zurückgegebenen Felder auswählen. gRPC ruft dagegen Methoden auf, die in einer Dienstdefinition stehen. Beide geben einen Teil der einheitlichen Schnittstelle zugunsten einer anderen Art von Datenverkehr auf.

Für eine HTTP-API ist ein großer Teil der einheitlichen Schnittstelle bereits vorhanden. HTTP definiert Methoden mit vereinbarter Bedeutung, Statuscodes und Header-Felder für Caching und bedingte Anfragen. Weil diese Bedeutung gemeinsam ist, kann ein Proxy oder Cache unterwegs auf eine Nachricht reagieren, ohne die konkrete API zu kennen. Eine API, die diese Teile wie vorgesehen verwendet, bekommt diesen Mechanismus ohne Zusatzarbeit. Gibt sie ihren Clients dagegen URL-Vorlagen zum Ausfüllen statt Links zum Folgen, verzichtet sie auf die Hypermedia-Bedingung. Ihre Clients hängen dann an der URL-Struktur des Servers: Ändert sie sich, funktionieren sie nicht mehr. Genau diese Kopplung soll die einheitliche Schnittstelle beseitigen.

Wie es in der Praxis aussieht

Der folgende Austausch legt eine Bestellung an und beantwortet die Anfrage mit der Adresse der neuen Ressource in `Location` und einem Entity-Tag in `ETag`. RFC 9110 verwendet selbst einen `ETag` in einer 201-Antwort als Beispiel: Er kann bei späteren bedingten Anfragen als Validator dienen, damit zwei Clients nicht unbemerkt die Änderungen des jeweils anderen überschreiben.

```
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"}
```

Jede Methode trägt eine Bedeutung, auf die sich jeder Vermittler verlassen darf. RFC 9110 definiert GET, HEAD, OPTIONS und TRACE als sicher: Der Client fordert keine Zustandsänderung beim Server an und erwartet auch keine. PUT und DELETE sind zusammen mit den sicheren Methoden idempotent. Mehrere identische Anfragen haben damit dieselbe beabsichtigte Wirkung wie eine einzelne. POST hat keine dieser beiden Eigenschaften, und RFC 5789 sagt dasselbe über PATCH. Diese Eigenschaften lassen einen Crawler Links folgen, ohne Schaden anzurichten, einen Cache vorab Daten abrufen und einen Client einen PUT wiederholen, dessen Verbindung vor Eintreffen der Antwort abbrach.

Eine REST-API lässt sich mit einem einfachen generischen HTTP-Client bedienen. Diese Website zeigt, was das spart und kostet. Als ein zweiter Server aus Code bereitgestellt wurde, wurden die Firewalls beider Cloud-Anbieter direkt über deren REST-APIs mit einer generischen HTTP-Aufgabe konfiguriert statt mit einer anbieterspezifischen Collection. Das beseitigt eine Abhängigkeit und auseinanderlaufende Versionen. Dafür muss die Form jeder Anfrage von Hand geschrieben werden. Dieselbe Seite nennt einen zweiten Preis: Die generische HTTP-Aufgabe deklariert keine Unterstützung für den Prüfmodus. Deshalb überspringt ein Probelauf des Plays, das die Maschine erstellt, jede seiner Aufgaben.

Wo es schiefgeht

Ein GET verändert etwas. Eine Adresse wie `/orders/7421/cancel`, die beim Abruf eine Bestellung storniert, bricht das Versprechen, dass GET sicher ist. RFC 9110 erklärt den Grund dieses Versprechens: Crawler und vorab abrufende Caches sollen ohne Angst vor Schäden arbeiten können. Der erste Crawler oder vorab abrufende Cache, der den Link findet, storniert die Bestellung. Legen Sie die Aktion stattdessen in die Methode, etwa als POST an die Bestellung oder PATCH ihres Status.

Jede Antwort ist 200 OK. Manche APIs geben für alles 200 zurück und legen das tatsächliche Ergebnis, Fehler oder nicht, in den Body. Die Nachrichten sind nicht mehr selbstbeschreibend: Caches, Proxys, Überwachung und Clientbibliotheken lesen den Statuscode, und alle sehen Erfolg. RFC 9457 definiert einen standardisierten Fehler-Body, `application/problem+json`, mit Feldern wie `type`, `title`, `status` und `detail`. Das Feld `status` ist nur ein Hinweis. Der Server muss denselben Statuscode in der HTTP-Antwort selbst senden, damit allgemeine HTTP-Software ohne Kenntnis dieses Formats richtig arbeitet.

Ein wiederholter POST erzeugt zwei Bestellungen. Bricht die Verbindung ab, nachdem der Server einen POST verarbeitet, der Client aber die Antwort noch nicht gelesen hat, weiß der Client nicht, ob die Bestellung existiert. Laut RFC 9110 sollte ein Client eine nicht idempotente Anfrage nicht automatisch wiederholen, außer er weiß, dass sie tatsächlich doch idempotent ist, oder kann feststellen, dass der erste Versuch nie angewandt wurde. Ein Proxy darf das überhaupt nicht tun. Die RFC merkt auch an, dass manche Clients den riskanteren Weg nehmen und einen POST trotzdem wiederholen, wenn sich die Verbindung vor dem ersten Teil der Antwort schließt, besonders bei einer gerade inaktiven persistenten Verbindung. Ein Ausweg ist, den Client die Kennung wählen zu lassen und die Bestellung per PUT an diese Adresse zu senden. PUT ist idempotent: Die erste Anfrage legt die Bestellung mit 201 an, eine Wiederholung ersetzt sie nur durch denselben Zustand.

Zwei Clients überschreiben einander. Beide lesen die Bestellung, ändern jeweils ein anderes Feld und senden das Ganze per PUT zurück. Die zweite Schreiboperation macht die erste unbemerkt rückgängig. RFC 9110 nennt dies das Lost-Update-Problem. Abhilfe schafft der Entity-Tag, den jeder Client beim Lesen erhielt und in `If-Match` zurücksendet. Der Server wendet die Änderung dann nur an, wenn die Ressource noch in dem Zustand ist, den dieser Client zuletzt gesehen hat. Andernfalls darf er sie nicht anwenden und kann dies mit 412 Precondition Failed mitteilen. Der Client liest die Bestellung dann erneut, trägt seine Änderung wieder ein und versucht es noch einmal.

```
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
```

Der Server erinnert sich an das Gespräch. Ein Server, der zwischen Anfragen Zustand je Client aufbewahrt, etwa die aktuelle Seite einer Liste oder ein halb ausgefülltes Formular im Speicher, hat die Bedingung der Zustandslosigkeit aufgegeben. Eine Anfrage ist für sich nicht mehr verständlich. Der Loadbalancer muss jeden Client zur selben Maschine zurückschicken, und ein Neustart verwirft den Zustand. Verloren gehen genau Fieldings Gründe für Zustandslosigkeit: Sichtbarkeit, leichtere Wiederherstellung nach Teilausfällen und ein Server, der

nach der Antwort Ressourcen freigeben kann. Legen Sie den Zustand in die Anfrage, etwa als Seitentoken in der URL, oder machen Sie das Unfertige zu einer eigenen Ressource wie einem Bestellentwurf, den jeder Server lesen kann.

Weiterführende Lektüre

- Fielding, Kapitel 5: Representational State Transfer, REST — Die Definition: jede Bedingung, die daraus entstehenden Eigenschaften und was dafür aufgegeben wird.
- RFC 9110: HTTP-Semantik — Methoden, Sicherheit und Idempotenz, Statuscodes und bedingte Anfragen mit `ETag` und `If-Match`.
- RFC 5789: Die PATCH-Methode für HTTP — Teilweise Änderung einer Ressource und warum PATCH weder sicher noch idempotent ist.
- RFC 9457: Problemdetails für HTTP-APIs — Ein standardisierter JSON-Body für Fehlerantworten, der die Bedeutung des Statuscodes erhält.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist REST?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/rest/