Engineer

Backend

der Teil eines Systems, den die Lesenden nicht herunterladen

Was es ist

Ein Backend ist der Teil eines Softwaresystems auf Rechnern unter Kontrolle des Betreibers. Es speichert Daten, wendet die Regeln dazu an, wer was tun darf, und beantwortet Anfragen von Browsern, Apps und anderen Diensten. Auch die Schreibweisen Back End und Back-End sind gebräuchlich. Derselbe Bereich heißt oft Serverseite.

In HTTP-Begriffen entspricht es dem Origin Server aus RFC 9110: „einem Programm, das maßgebliche Antworten für eine bestimmte Zielressource erzeugen kann“. Die W3C-Architektur des World Wide Web ergänzt die Lesersicht: Eine URI identifiziert eine Ressource. Zum Browser reist ihre Repräsentation, also Daten, die Informationen über den Zustand der Ressource kodieren. Ein Kunde sieht eine Bestellseite. Das Backend hält Bestellung, Bestand und Zahlung und entscheidet über die Antwort.

Meist stellt es eine API bereit: HTTP-Endpunkte für Frontend, mobile App oder ein fremdes Unternehmenssystem. Die API ist ein Vertrag über Pfade, Eingaben, Rückgaben und mögliche Fehler. Die OpenAPI Specification ist die übliche programmiersprachenunabhängige Beschreibung dafür. Werkzeuge erzeugen daraus Clients, Dokumentation und Tests. Hinter der API verteilen sich Aufgaben meist auf Anfragebearbeitung, Geschäftslogik und Speicherung.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Backend?“ 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

Zuerst ist zu klären, ob zur Anfragezeit überhaupt ein Backend laufen muss. Eine für alle gleiche Seite kann einmal generiert und als Datei geliefert werden. Ein laufendes Backend rechtfertigt sich, wenn die Anfrage etwas ändert oder vom Absender abhängt: Bestellung, Anmeldung oder Suche in privaten Daten. Es trägt auch dauerhafte Kosten. Software, die Internetanfragen beantwortet, muss gesichert, gepatcht und bezahlt werden.

Die zweite Frage ist Vertrauen. Auf kontrollierten Rechnern ist das Backend der Ort für verbindliche Regeln. Jede Frontendprüfung lässt sich mit direkten Anfragen umgehen. Preis, Berechtigung und jede andere wichtige Regel werden deshalb auf dem Server gegen angemeldete Person und konkreten Datensatz geprüft. Der OWASP Application Security Verification Standard macht daraus umsetzbare und testbare Anforderungen.

Die dritte Frage ist der Ort der Logik. Eine auf Handler, Serviceklasse und Datenbanktrigger verteilte Regel hat drei Änderungsstellen und drei auseinanderlaufende Kopien. Manche Teams halten Regeln im Anwendungscode und verwenden die Datenbank als Speicher. Andere verlagern Regeln und Datenformung in die Datenbank und halten die API dünn. Beides kann funktionieren, solange jede Regel genau ein Zuhause hat.

Viertens geht es um den Clientvertrag. Jede HTTP-Antwort erhält einen Statuscode mit definierter Bedeutung, nach dem Clients, Proxys und Überwachung handeln. 422 Unprocessable Content besagt, dass die Anfrage syntaktisch korrekt ist und ein unterstütztes Format hat, inhaltlich aber semantisch falsch ist, etwa eine Buchung in der Vergangenheit. RFC 9457 ergänzt mit Problem Details ein standardisiertes Fehlerformat: ein kleines JSON-Objekt mit Typ, Titel, Status, Detail und Instanz, ausgeliefert als `application/problem+json`. Clients unterscheiden Fehler damit ohne Textanalyse.

Wie es in der Praxis aussieht

Gewöhnlich führt ein Staffellauf von Umformungen von der Datenbank zum Browser. Die API liest Zeilen in Strukturen, formt sie um und serialisiert sie wieder als JSON. Auf einer Plattform lässt das Backend die Datenbank jede Antwort bauen und reicht sie unverändert weiter. PostgreSQL-Funktionen setzen die gesamte JSON-Antwort direkt bei den Daten zusammen. Der Go-Handler erhält `json.RawMessage` und leitet es unverändert weiter. Ein kleiner `QueryJSON`-Helfer machte das zum einfachsten Weg. DTOs, Mapper und Antwortstrukturen entfielen, Handler wurden erheblich kürzer. Änderungen einer Funktionsrückgabe fließen ohne Handleränderung bis zum Client.

Eingehende Anfragen erhalten dieselbe Behandlung aus einer Quelle über einen Vertrag, der bei der Datenbank beginnt und nach außen führt. Schema und Funktionen bestimmen die Datenformen, Go-Typen die API. Huma erzeugt daraus OpenAPI, woraus der Frontendclient mit 44.076 Zeilen typisiertem TypeScript generiert wird. Formulargrenzen stammen aus derselben Quelle, statt in einen Validator abgeschrieben zu werden. Jeder Übergang besitzt eine Prüfung. Ein nur einseitig geändertes Feld scheitert im Build, Minuten nach der Änderung. Der Preis ist die Neuerzeugung: Jede Änderung einer Backend-Datenform muss durch erneutes Generieren aller folgenden Glieder weitergetragen werden.

Im dokumentierten Entwurfsstadium traf diese Website die gegenteilige Wahl. Sie wurde vollständig beim Build erzeugt und benötigte kein Anwendungsbackend für eingehende Anfragen. Ihre vier Anforderungen, darunter drei Sprachen und der Verzicht auf einen separat abgerechneten Anwendungsdienst, führten zu dieser Wahl: Ein Anwendungsbackend zur Laufzeit hätte abgesichert, gepatcht und auf einer Datenschutzseite erklärt werden müssen. Abgesehen von der dokumentierten, eng begrenzten JavaScript-Ausnahme für einen Portfoliofilter mussten interaktive Funktionen beim Build gelöst werden oder entfallen. Das beschreibt den dokumentierten Entwurfsstand, nicht den aktuellen Bestand oder die aktuelle Implementierung der Website; es bedeutet nicht, dass das Hosting kostenlos war.

Wo es schiefgeht

Backendfehler bleiben oft still, bis eine ungeplante Anfrage eintrifft.

Ein Nutzer ändert fremde Daten. Der Endpunkt prüft die Anmeldung und lädt dann den Datensatz zu jeder in der URL genannten Kennung. Aus `/invoices/1041` wird `/invoices/1042`, mehr braucht es nicht. Prüfen Sie Eigentum oder Berechtigung serverseitig für jedes berührte Objekt an einer gemeinsamen Stelle aller Handler. Behalten Sie einen Test bei, der den Datensatz eines anderen Kontos anfordert und 403 oder 404 erwartet.

Eine Handlung geschieht zweimal. Ein Link zum Abbrechen, der eine GET-Anfrage verwendet, wird von einem Crawler oder vorladenden Browser verfolgt. Die Handlung läuft ohne Klick. RFC 9110 definiert GET, HEAD, OPTIONS und TRACE als sicher, damit Crawler und Vorladen ohne Schadensgefahr funktionieren. Der andere Weg ist die Wiederholung: Eine Zahlung läuft ins Zeitlimit, der Client sendet erneut, der Kunde zahlt doppelt. Legen Sie Zustandsänderungen hinter POST, PUT, PATCH oder DELETE. Wiederholbare Schreibvorgänge erhalten einen vom Server gemerkten Schlüssel, sodass die zweite Kopie das erste Ergebnis zurückgibt.

Jeder Fehler wird als 500 oder als 200 mit Fehlertext gemeldet. Ein Sammelhandler behandelt alles gleich. Die Überwachung zählt Tippfehler als Ausfälle, Wiederholungslogik sendet stets erfolglose Anfragen erneut. Ordnen Sie Fehler ihrem Status zu: 400 oder 422 für ungültige Eingaben, 401 und 403 für Identität und Rechte, 404 für fehlende Ressourcen, 409 für Konflikte und 5xx nur für Serverfehler. Details gehen als Problem Details hinaus:

```
{
    "type": "https://example.com/problems/booking-in-past",
    "title": "Booking date is in the past",
    "status": 422,
    "detail": "The date 2026-10-01 is before today.",
    "instance": "/bookings/8812"
}
```

RFC 9457 sagt, Problem Details seien „kein Debugging-Werkzeug für die zugrunde liegende Implementierung“. Stacktraces und SQL bleiben deshalb im Serverprotokoll.

Mit wachsenden Daten werden Seiten langsam. Ein Handler lädt Bestellungen und fragt danach für jede einzelne den zugehörigen Kunden ab. 50 Bestellungen erzeugen 51 einzelne Aufrufe der Datenbank. Mit zehn Entwicklungszeilen wirkt

es schnell, mit zehntausend im Betrieb wird es zäh. Laden Sie Beziehungen gemeinsam per Join oder lassen Sie die Datenbank die ganze Antwort bauen. Beobachten Sie die Abfragen pro Anfrage genauso genau wie die Antwortzeit.

Weiterführende Lektüre

- Architecture of the World Wide Web, Volume One — die W3C-Empfehlung zu Ressourcen, Repräsentationen und sicheren beziehungsweise unsicheren Interaktionen.
- RFC 9110: HTTP Semantics — der IETF-Standard für Methoden, Statuscodes und sichere oder idempotente Anfragen.
- OpenAPI Specification — das sprachunabhängige HTTP-API-Format für Menschen und Codegeneratoren.
- RFC 9457: Problem Details for HTTP APIs — die Standard-JSON-Form maschinenlesbarer Fehlerantworten.
- OWASP Application Security Verification Standard — Sicherheitsanforderungen zum Bauen und Testen von Webanwendungen.

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 ein Backend?“ 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/backend/