Engineer

Backend

den halva av ett system som läsaren aldrig laddar

Vad det är

En backend är den del av ett mjukvarusystem som körs på maskiner som operatören själv kontrollerar: den lagrar data, tillämpar reglerna om vem som får göra vad och besvarar de förfrågningar som kommer från webbläsare, appar och andra tjänster. Ordet skrivs också back end eller back-end, och samma område kallas ofta serversidan.

I HTTP:s termer är en backend det som RFC 9110 kallar en ursprungsserver (origin server), ett program som kan skapa auktoritativa svar för en given målresurs. W3C:s Architecture of the World Wide Web lägger till läsarens perspektiv: en URI identifierar en resurs, och det som färdas till webbläsaren är en representation av den, data som kodar information om resursens tillstånd. En kund ser en sida med en beställning; backenden håller beställningen, lagersaldot och betalningen och avgör vad som ska ingå i svaret.

En backend exponerar oftast ett API, en uppsättning HTTP-endpoints som en frontend, en mobilapp eller ett annat företags system kan anropa. API:et är ett kontrakt: vilka sökvägar som finns, vad var och en tar emot, vad den returnerar och vilka fel den kan ge. OpenAPI Specification är det vanliga sättet att skriva ned det kontraktet, som en beskrivning oberoende av programspråk ur vilken verktyg kan generera klienter, dokumentation och tester. Bakom API:et delas arbetet oftast upp i lager: hantering av förfrågningar, affärslogik och lagring.

Använd begreppet i ett beslut

Använd ”Vad är en backend?” 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

Den första frågan är om ett system alls behöver en backend som körs i det ögonblick förfrågan kommer. En sida som är likadan för varje besökare kan genereras en gång och serveras som filer. En backend som körs förtjänar sin plats när en förfrågan måste ändra något eller beror på vem som skickade den: en beställning, en inloggning, en sökning i privata data. Den bär också en löpande kostnad, eftersom mjukvara som besvarar förfrågningar från internet måste säkras, patchas och betalas för.

Den andra är tillit. Eftersom backenden körs på maskiner som operatören kontrollerar är den det enda stället där en regel kan upprätthållas. Allt som frontenden kontrollerar kan kringgås av någon som skickar förfrågningar direkt, så varje regel som spelar roll, från ett pris till en behörighet, kontrolleras på servern mot den inloggade användaren och den specifika posten. OWASP Application Security Verification Standard gör det till krav som ett team kan bygga efter och testa mot.

Den tredje är var logiken bor. En regel som är utspridd över en handler, en tjänsteklass och en trigger i databasen har tre ställen att ändra och tre kopior som kan glida isär. Vissa team håller reglerna i applikationskoden och använder databasen som lagring; andra flyttar reglerna och formningen av data in i databasen och håller API:et tunt. Båda kan fungera, så länge varje regel har ett enda hem.

Den fjärde är kontraktet med klienterna. HTTP ger varje svar en statuskod med en definierad betydelse, och klienter, proxyer och verktyg för övervakning agerar på den. En 422 Unprocessable Content säger att förfrågan var syntaktiskt korrekt och i ett format som stöds, men semantiskt fel, t.ex. en bokning daterad i det förflutna. RFC 9457 lägger till en standardiserad kropp för fel, kallad problem details: ett litet JSON-objekt med fälten type, title, status, detail och instance, som skickas som `application/problem+json`, så att en klient kan skilja ett fel från ett annat utan att tolka löptext.

Så ser det ut i praktiken

Hur mycket en backend ska göra med data på vägen, och om den ska köras över huvud taget, är beslut som har fattats åt olika håll i arbetet nedan.

På en plattform, i rollen som plattformsarkitekt, låter backenden databasen bygga varje svar och skickar det vidare orört. Sidan kallar den vanliga vägen från databas till webbläsare en stafett av transformationer: API:et deserialiserar rader till structar, formar om dem och serialiserar dem tillbaka till JSON. Här sätter PostgreSQL-funktioner ihop hela svaret som JSON, där datan redan finns, och Go-handlern tar emot det som `json.RawMessage` och vidarebefordrar det orört. En liten hjälpfunktion, `QueryJSON`, gjorde det till minsta motståndets väg. DTO:erna, mappningsfunktionerna och svars-structarna föll bort, handlerkoden blev dramatiskt kortare, och en ändring av vad en funktion returnerar flödar rakt igenom till klienten utan att någon ändrar handlerkod.

En annan sida, från rollen som backendingenjör, tillämpar samma idé om en enda källa på de förfrågningar som kommer in, med ett kontrakt som börjar i databasen och går utåt. Databasschemat och dess funktioner definierar strukturerna, Go-typerna definierar API:et, Huma genererar OpenAPI-beskrivningen utifrån dem, och en typad TypeScript-klient på 44 076 rader genereras från den beskrivningen. De gränser som gränssnittet upprätthåller i ett formulär kommer från samma ställe i stället för att skrivas in på nytt i en validerare. Varje steg har ett skydd, så ett fält kan inte ändras på bara den ena sidan: avvikelsen fallerar i ett bygge, minuter efter ändringen. Priset är att varje ändring av en datastruktur som backenden definierar måste föras vidare genom kedjan genom att varje efterföljande led genereras på nytt.

I det dokumenterade designskedet gjorde den här webbplatsen det motsatta valet. Den genererades helt vid byggtid och behövde ingen applikationsbackend för att behandla anrop. Dess fyra krav, däribland att fungera på tre språk och undvika en separat fakturerad applikationstjänst, ledde till valet: en applikationsbackend som kördes på servern skulle behöva säkras, patchas och förklaras på en integritetssida. Utöver det dokumenterade, avgränsade JavaScript-undantaget för ett portfoliofilter behövde interaktiva funktioner lösas vid byggtid eller utelämnas. Detta beskriver det dokumenterade designskedet, inte webbplatsens nuvarande innehåll eller implementering; det betyder inte att hosting saknade kostnader.

Var det går fel

Det som brister i en backend brukar hålla tyst tills det kommer en förfrågan som ingen hade planerat för.

En användare ändrar en post som tillhör någon annan. Endpointen kontrollerar att anroparen är inloggad och laddar sedan det id som URL:en anger, så det räcker att ändra `/invoices/1041` till `/invoices/1042` i en förfrågan. Kontrollera ägarskap eller behörighet på servern för varje objekt som en förfrågan rör, på ett ställe som varje handler passerar, och behåll ett test som ber om ett annat kontos post och förväntar sig 403 eller 404.

En åtgärd sker två gånger. En länk som avbryter något via en GET-förfrågan följs av en sökrobot eller en webbläsare som hämtar sidor i förväg, och åtgärden körs utan att någon har klickat. RFC 9110 definierar GET, HEAD, OPTIONS och TRACE som säkra just för att sökrobotar och förhämtning ska kunna arbeta utan risk för att orsaka skada. Den andra vägen är ett nytt försök: en betalningsförfrågan når sin tidsgräns, klienten skickar den igen och kunden betalar två gånger. Lägg varje ändring av tillstånd bakom POST, PUT, PATCH eller DELETE, och ge skrivningar som kan göras om en nyckel som servern minns, så att den andra kopian returnerar resultatet av den första och skrivningen blir idempotent.

Varje fel kommer tillbaka som 500, eller som 200 med ett felmeddelande inuti. En enda handler som fångar allt rapporterar varje misslyckande på samma sätt, så övervakningen räknar skrivfel som avbrott och logiken för nya försök upprepar förfrågningar som kommer att misslyckas varje gång. Koppla varje fel till den statuskod som beskriver det: 400 eller 422 för felaktig indata, 401 och 403 för identitet och behörighet, 404 för en resurs som saknas, 409 för en konflikt och 5xx bara för serverns egna fel. Skicka detaljerna som problem details:

```
{
    "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 säger att problem details inte är ett verktyg för att felsöka den underliggande implementationen, så stackspår och SQL stannar i serverns loggar.

Sidorna blir långsammare när datan växer. En handler laddar en lista med beställningar och frågar sedan efter kunden för var och en, så en sida med 50 beställningar gör 51 rundturer till databasen. Det känns snabbt under utvecklingen med tio rader och segt i produktion med tio tusen. Hämta relaterade data i en enda databasfråga med en join, eller låt databasen sätta ihop hela svaret, och följ antalet databasfrågor per förfrågan lika noga som svarstiden.

Läs mer

- Architecture of the World Wide Web, Volume One — W3C:s rekommendation om resurser, representationer och säkra och osäkra interaktioner.
- RFC 9110: HTTP Semantics — IETF:s standard för metoder, statuskoder och vad som gör en förfrågan säker eller idempotent.
- OpenAPI Specification — det språkoberoende formatet för att beskriva ett HTTP-API, som läses av både människor och kodgeneratorer.
- RFC 9457: Problem Details for HTTP APIs — den standardiserade JSON-formen för maskinläsbara felsvar.
- OWASP Application Security Verification Standard — en lista över säkerhetskrav för att bygga och testa webbapplikationer.

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 en backend?” 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/backend/