Engineer

Backend

den halvdelen av et system leseren aldri laster inn

Hva det er

En backend er den delen av et programvaresystem som kjører på maskiner operatøren selv kontrollerer: den lagrer dataene, håndhever reglene for hvem som kan gjøre hva, og svarer på forespørslene som kommer fra nettlesere, apper og andre tjenester. Den skrives også back end eller back-end, og det samme området kalles ofte serversiden.

I HTTP-termer er en backend det RFC 9110 kaller en opprinnelsesserver (origin server), et program som kan lage autoritative svar for en gitt målressurs. W3Cs Architecture of the World Wide Web legger til leserens perspektiv: en URI identifiserer en ressurs, og det som reiser til nettleseren, er en representasjon av den, data som koder informasjon om tilstanden til ressursen. En kunde ser en side med en bestilling; backenden holder på bestillingen, lagerbeholdningen og betalingen, og avgjør hva som skal med i svaret.

En backend eksponerer som regel et API, et sett HTTP-endepunkter som en frontend, en mobilapp eller et annet selskaps system kan kalle. API-et er en kontrakt: hvilke stier som finnes, hva hver av dem tar imot, hva den returnerer, og hvilke feil den kan gi. OpenAPI-spesifikasjonen er den vanlige måten å skrive ned kontrakten på, som en beskrivelse uavhengig av programmeringsspråk, og verktøy kan generere klienter, dokumentasjon og tester fra den. Bak API-et er arbeidet som regel delt i lag: håndtering av forespørsler, forretningslogikk og lagring.

Bruk begrepet i en beslutning

Bruk «Hva er en backend?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Delen brukerne aldri ser: hvor dataene ligger, hvilke regler som gjelder, og hvordan andre systemer får lov til å snakke med den.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Designet og driftet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tvers av tre databasemotorer

Designet og driftet 6 ETL/ELT-pipelines Ta kontakt →

Hvorfor det er viktig

Det første spørsmålet er om et system i det hele tatt trenger en backend som kjører idet forespørselen kommer. En side som er lik for hver besøkende, kan genereres én gang og serveres som filer. En backend i drift fortjener plassen sin når en forespørsel må endre noe eller avhenger av hvem som sendte den: en bestilling, en innlogging, et søk i private data. Den har også en fast kostnad, fordi programvare som svarer på forespørsler fra internett, må sikres, oppdateres og betales for.

Det andre er tillit. Fordi backenden kjører på maskiner operatøren kontrollerer, er den det eneste stedet en regel kan håndheves. Alt frontenden sjekker, kan hoppes over av noen som sender forespørsler direkte, så hver regel som betyr noe, fra en pris til en tillatelse, sjekkes på serveren mot den innloggede brukeren og den konkrete posten. OWASP Application Security Verification Standard gjør dette om til krav et team kan bygge etter og teste mot.

Det tredje er hvor logikken bor. En regel som er spredt over en handler, en tjenesteklasse og en trigger i databasen, har tre steder som må endres, og tre kopier som kan gli fra hverandre. Noen team holder reglene i applikasjonskoden og bruker databasen til lagring; andre skyver reglene og formingen av data inn i databasen og holder API-et tynt. Begge deler kan virke, så lenge hver regel har ett hjem.

Det fjerde er kontrakten med klientene. HTTP gir hvert svar en statuskode med en definert betydning, og klienter, proxyer og verktøy for overvåking handler ut fra den. En 422 Unprocessable Content sier at forespørselen var syntaktisk riktig og i et format som støttes, men semantisk feil, f.eks. en booking datert tilbake i tid. RFC 9457 legger til en standardisert kropp for feil, kalt problem details: et lite JSON-objekt med `type`, `title`, `status`, `detail` og `instance`, sendt som `application/problem+json`, slik at en klient kan skille én feil fra en annen uten å tolke fritekst.

Slik ser det ut i praksis

Hvor mye en backend skal gjøre, er et valg, og arbeidet her spenner fra en backend som knapt rører dataene, til ingen backend i det hele tatt ved kjøretid.

På én plattform lar backenden databasen bygge hvert svar og sender det videre urørt. Siden kaller den vanlige veien fra database til nettleser en stafett av transformasjoner: API-et pakker ut rader i structs, omformer dem og serialiserer dem tilbake til JSON. Her setter PostgreSQL-funksjoner sammen hele svaret som JSON, der dataene allerede ligger, og Go-handleren tar det imot som `json.RawMessage` og sender det videre urørt. En liten `QueryJSON`-hjelper gjorde det til minste motstands vei. DTO-ene, mapperne og respons-structene falt bort, handlerkoden ble dramatisk kortere, og en endring i hva en funksjon returnerer, flyter rett gjennom til klienten uten at noen redigerer handlerkode.

En annen side bruker den samme ideen om én kilde på forespørslene som kommer inn, med en kontrakt som starter i databasen og går utover. Skjemaet og funksjonene i det definerer formene, Go-typene definerer API-et, Huma lager OpenAPI-beskrivelsen ut fra dem, og en typet TypeScript-klient på 44 076 linjer genereres fra den beskrivelsen. Grensene brukergrensesnittet håndhever i et webskjema, kommer fra samme sted i stedet for å bli skrevet inn på nytt i en validator. Hvert ledd har en vakt, så et felt kan ikke endres på bare én side: avviket feiler i et bygg, minutter etter endringen. Kostnaden er at hver endring i en datastruktur som backenden definerer, må føres videre gjennom kjeden ved å generere hvert etterfølgende ledd på nytt.

På det dokumenterte designstadiet tok dette nettstedet det motsatte valget. Det ble generert i sin helhet ved byggetid og trengte ingen applikasjonsbackend for å behandle forespørsler. De fire kravene, blant dem å virke på tre språk og unngå en separat fakturert applikasjonstjeneste, førte til valget: en applikasjonsbackend ved kjøretid måtte sikres, patches og forklares på en personvernside. Utover det dokumenterte, avgrensede JavaScript-unntaket for et portfoliofilter måtte interaktive funksjoner løses ved byggetid eller utelates. Dette beskriver det dokumenterte designstadiet, ikke nettstedets nåværende innhold eller implementering; det betyr ikke at hosting var kostnadsfritt.

Hvor det går galt

Feil i en backend holder seg gjerne stille til det kommer en forespørsel ingen hadde planlagt for.

En bruker endrer en post som tilhører noen andre. Endepunktet sjekker at den som kaller, er logget inn, og laster så den ID-en URL-en oppgir, så alt som skal til, er å endre `/invoices/1041` til `/invoices/1042` i en forespørsel. Sjekk eierskap eller tillatelse på serveren for hvert objekt en forespørsel rører, på ett sted hver handler går gjennom, og ha en test som ber om en annen kontos post og venter et 403 eller et 404.

En handling skjer to ganger. En lenke som avbestiller noe gjennom en GET-forespørsel, blir fulgt av en søkerobot eller en nettleser som henter sider på forhånd, og handlingen kjører uten at noen har klikket. RFC 9110 definerer GET, HEAD, OPTIONS og TRACE som trygge nettopp for at søkeroboter og forhåndshenting skal kunne virke uten frykt for å gjøre skade. Den andre veien er et nytt forsøk: en betalingsforespørsel får tidsavbrudd, klienten sender den på nytt, og kunden betaler to ganger. Legg hver endring av tilstand bak POST, PUT, PATCH eller DELETE, og gi skrivinger som kan bli forsøkt på nytt, en nøkkel serveren husker, slik at den andre kopien returnerer resultatet av den første.

Hver feil kommer tilbake som 500, eller som 200 med en feilmelding inni. Én samlehandler rapporterer hver feil på samme måte, så overvåkingen teller skrivefeil som driftsavbrudd, og logikken for nye forsøk gjentar forespørsler som kommer til å feile hver gang. Knytt hver feil til statuskoden som beskriver den: 400 eller 422 for ugyldige inndata, 401 og 403 for identitet og tillatelse, 404 for en ressurs som mangler, 409 for en konflikt, og 5xx bare for serverens egne feil. Send detaljene 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 sier at problem details ikke er et feilsøkingsverktøy for den underliggende implementasjonen, så stakkspor og SQL blir værende i serverens logger.

Sidene blir tregere etter hvert som dataene vokser. En handler laster en liste med bestillinger og spør så etter kunden for hver av dem, så en side med 50 bestillinger gir 51 rundturer til databasen. Det føles raskt under utvikling med ti

rader og sleper seg av gårde i produksjon med ti tusen. Hent relaterte data i én spørring med en join, eller la databasen sette sammen hele svaret, og følg med på antall spørringer per forespørsel like nøye som på svartiden.

Les mer

- Architecture of the World Wide Web, Volume One — W3C-anbefalingen om ressurser, representasjoner og trygge og utrygge interaksjoner.
- RFC 9110: HTTP Semantics — IETF-standarden for metoder, statuskoder og hva som gjør en forespørsel trygg eller idempotent.
- OpenAPI Specification — det språkuavhengige formatet for å beskrive et HTTP-API, lest av både mennesker og kodegeneratorer.
- RFC 9457: Problem Details for HTTP APIs — standardformen for feilsvar i JSON som maskiner kan lese.
- OWASP Application Security Verification Standard — en liste med sikkerhetskrav for å bygge og teste webapplikasjoner.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en backend?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/backend/