Engineer

Backend

den halvdel af et system læseren aldrig henter

Hvad det er

En backend er den del af et softwaresystem, der kører på maskiner, som dens operatør selv styrer: den gemmer data, anvender reglerne for, hvem der må hvad, og besvarer de forespørgsler, der kommer fra browsere, apps og andre tjenester. Det skrives også back end eller back-end, og det samme område kaldes ofte server side eller serversiden.

Med HTTP's begreber er en backend det, RFC 9110 kalder en origin server: et program, der kan frembringe autoritative svar for en given target resource. W3C's Architecture of the World Wide Web tilføjer læserens synsvinkel. En URI identificerer en ressource, og det, der rejser til browseren, er en repræsentation af den, data, der koder information om ressourcens tilstand. En kunde ser en ordreside, mens backenden har ordren, lagerbeholdningen og betalingen og afgør, hvad der kommer med i svaret.

En backend udstiller som regel et API, et sæt HTTP-endpoints, som en frontend, en mobilapp eller et andet firmas system kan kalde. API'et er en kontrakt: hvilke stier der findes, hvad hver af dem tager imod, hvad den returnerer, og hvilke fejl den kan give. OpenAPI Specification er den gængse måde at skrive kontrakten ned på, som en beskrivelse, der er uafhængig af programmeringssprog, og som værktøjer kan generere klienter, dokumentation og tests ud fra. Bag API'et er arbejdet som regel delt i lag: håndtering af forespørgsler, forretningslogik og lagring.

Brug begrebet i en beslutning

Brug »Hvad er en backend?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Den del, brugerne aldrig ser: hvor data ligger, hvilke regler der gælder, og hvordan andre systemer må tale med den.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har designet og drevet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tre databasemotorer

Har designet og drevet 6 ETL/ELT-pipelines Kontakt os →

Hvorfor det betyder noget

Det første spørgsmål er, om et system overhovedet har brug for en backend, der kører, når forespørgslen kommer. En side, der er ens for alle besøgende, kan genereres én gang og serveres som filer. En kørende backend gør sig fortjent til sin plads, når en forespørgsel skal ændre noget eller afhænger af, hvem der har sendt den: en ordre, et login, en søgning i private data. Den har også en løbende pris, for software, der besvarer forespørgsler fra internettet, skal sikres, opdateres og betales.

Det andet er tillid. Fordi backenden kører på maskiner, operatøren styrer, er den det sted, en regel kan håndhæves. Alt, hvad frontenden tjekker, kan springes over af en, der sender forespørgsler direkte, så hver regel, der betyder noget, fra en pris til en tilladelse, tjekkes på serveren mod den indloggede bruger og den konkrete post. OWASP Application Security Verification Standard gør det til krav, et team kan bygge efter og teste imod.

Det tredje er, hvor logikken bor. En regel, der er spredt over en handler, en serviceklasse og en databasetrigger, har tre steder at blive ændret og tre kopier, der kan glide fra hinanden. Nogle teams holder reglerne i applikationskoden og bruger databasen til lagring; andre lægger reglerne og formningen af data ind i databasen og holder API'et tyndt. Begge dele kan fungere, så længe hver regel har ét hjem.

Det fjerde er kontrakten med klienterne. HTTP giver hvert svar en statuskode med en defineret betydning, og klienter, proxyer og overvågningsværktøjer handler ud fra den. Et 422 Unprocessable Content siger, at forespørgslen var syntaktisk korrekt og i et understøttet format, men semantisk forkert, for eksempel en booking dateret i fortiden. RFC 9457 tilføjer et standardformat til fejlsvar, kaldet problem details: et lille JSON-objekt med en type, en title, en status, en detail og en instance, sendt som `application/problem+json`, så en klient kan skelne én fejl fra en anden uden at fortolke prosa.

Sådan ser det ud i praksis

På én platform lader backenden databasen bygge hvert svar og sender det videre uændret. Siden kalder den sædvanlige vej fra database til browser et stafetløb af transformationer: API'et unmarshaller rækker til structs, omformer dem og serialiserer dem tilbage til JSON. Her samler PostgreSQL-funktioner hele svaret som JSON, der hvor dataene allerede ligger, og Go-handleren tager imod det som `json.RawMessage` og videresender det, som det er. En lille `QueryJSON`-helper gjorde det til den letteste vej. DTO'erne, mapperne og svar-structene faldt bort, handler-koden blev markant kortere, og en ændring i det, en funktion returnerer, løber direkte igennem til klienten, uden at nogen retter i handler-koden.

Den samme idé om én kilde kan også styre de forespørgsler, der kommer ind, gennem en kontrakt, der starter ved databasen og løber udad. Skemaet og dets funktioner definerer datastrukturerne, Go-typerne definerer API'et, Huma genererer OpenAPI-beskrivelsen ud fra dem, og en typet TypeScript-klient på 44.076 linjer genereres ud fra den beskrivelse. De grænser, brugerfladen håndhæver i en formular, hentes fra samme kilde og skrives ikke ind i en validator en gang til. Hvert led er bevogtet, så et felt ikke kan ændre sig på kun den ene side: uoverensstemmelsen får et build til at fejle, minutter efter ændringen. Prisen er, at hver ændring af en datastruktur, som backenden definerer, skal føres videre gennem kæden ved at generere hvert efterfølgende led på ny.

På det dokumenterede designtrin traf dette site det modsatte valg. Det blev genereret udelukkende på byggetidspunktet og havde ikke brug for en applikationsbackend til at behandle forespørgsler. Dets fire krav, herunder at virke på tre sprog og undgå en særskilt faktureret applikationstjeneste, førte til valget: en applikationsbackend i drift skulle sikres, opdateres og forklares på en privatlivsside. Bortset fra den dokumenterede, afgrænsede JavaScript-undtagelse til et portfoliofilter skulle interaktive funktioner løses på byggetidspunktet eller udelades. Dette beskriver det dokumenterede designtrin, ikke sitets aktuelle indhold eller implementering; det betyder ikke, at hosting var uden omkostninger.

Hvor det går galt

Fejl i en backend holder sig som regel skjult, indtil der kommer en forespørgsel, som ingen havde planlagt.

En bruger ændrer en post, der tilhører en anden. Endpointet tjekker, at den, der kalder, er logget ind, og indlæser så det ID, URL'en nævner, så det er nok at rette `/invoices/1041` til `/invoices/1042` i en forespørgsel. Tjek ejerskab eller tilladelse på serveren for hvert objekt, en forespørgsel rører, ét sted, som alle handlere går igennem, og hav en test, der beder om en anden kontos post og forventer et 403 eller et 404.

En handling sker to gange. Et link, der annullerer noget via en GET-forespørgsel, bliver fulgt af en crawler eller af en browser, der henter sider på forhånd, og handlingen udføres, uden at nogen har klikket. RFC 9110 definerer GET, HEAD, OPTIONS og TRACE som safe-metoder, så spiders og pre-fetching kan arbejde uden frygt for at gøre skade. Den anden vej er et genforsøg: en betalingsforespørgsel løber ud på tid, klienten sender den igen, og kunden betaler to gange. Læg enhver ændring af tilstand bag POST, PUT, PATCH eller DELETE, og giv skrivninger, der kan blive gentaget, en nøgle, som serveren husker, så den anden kopi returnerer resultatet af den første.

Hver fejl kommer tilbage som et 500 eller som et 200 med en fejlbesked indeni. Én catch-all-handler rapporterer alle fejl på samme måde, så overvågningen tæller tastefejl som nedbrud, og logikken for genforsøg gentager forespørgsler, der vil fejle hver gang. Knyt hver fejl til den statuskode, der beskriver den: 400 eller 422 for dårligt input, 401 og 403 for identitet og tilladelse, 404 for en ressource, der mangler, 409 for en konflikt og kun 5xx for serverens egne fejl. Send detaljerne 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 siger, at problem details ikke er et fejlfindingsværktøj til den underliggende implementering, så stacktraces og SQL bliver i serverens logs.

Sider bliver langsommere, efterhånden som dataene vokser. En handler indlæser en liste med ordrer og spørger derefter efter kunden til hver af dem, så en side med 50 ordrer giver 51 rundture til databasen. Det føles hurtigt i

udvikling med ti rækker og trægt i produktion med ti tusind. Hent relaterede data i én forespørgsel med et join, eller lad databasen samle hele svaret, og hold lige så nøje øje med antallet af databasekald pr. forespørgsel som med svartiden.

Læs mere

- Architecture of the World Wide Web, Volume One — W3C-anbefalingen om ressourcer, repræsentationer og sikre og usikre interaktioner.
- RFC 9110: HTTP Semantics — IETF-standarden for metoder, statuskoder og det, der gør en forespørgsel safe eller idempotent.
- OpenAPI Specification — det sproguafhængige format til at beskrive et HTTP-API, som både mennesker og kodegeneratorer læser.
- RFC 9457: Problem Details for HTTP APIs — standardformen for maskinlæsbare fejlsvar i JSON.
- OWASP Application Security Verification Standard — en liste med sikkerhedskrav til at bygge og teste webapplikationer.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en backend?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/backend/