Engineer

Frontend

systemdelen, der står for præsentation og interaktion

Hvad det er

En frontend er den del af et softwaresystem, som folk ser på og betjener: sider, skærmbilleder, formularer og knapper, sammen med den kode, der tegner dem og reagerer på input. Det skrives også front end eller front-end, og på nettet kaldes det samme område ofte client side eller klientsiden.

På nettet bygges en frontend af tre sprog, som en browser forstår. HTML giver en side struktur og betydning, CSS bestemmer, hvordan den ser ud, og JavaScript tilføjer adfærd. Browseren er det, W3C's Architecture of the World Wide Web kalder en user agent: software, der handler på vegne af en person, henter data fra servere og præsenterer dem. Alt, hvad en frontend sender til den, ender på en enhed, der tilhører en anden.

Dens makker er backenden, der gemmer data, anvender forretningsreglerne og besvarer forespørgsler. De to taler sammen over HTTP, som regel gennem et API. Det er fristende at trække grænsen mellem dem, hvor netværket går, men nutidens frameworks udvisker den. I Next.js er sider og layouts som standard Server Components: de frembringer det, brugeren ser, og kører på en server. Et `"use client"` -direktiv øverst i en fil markerer grænsen, og filen, dens imports og de komponenter, den renderer, kommer med i det bundle, der sendes til browseren. Det, der gør kode til frontend, er dens opgave, præsentation og interaktion, og hvor koden kører, er en separat beslutning.

Brug begrebet i en beslutning

Brug »Hvad er en frontend?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Brugerflader, der indlæses hurtigt, virker på en telefon og opfører sig ens i alle de browsere, jeres kunder nu bruger.

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 styret full-stack-udviklingen af GIS-kort — full stack-kortløsning, fra database til browser

Har styret full-stack-udviklingen af GIS-kort Kontakt os →

Hvorfor det betyder noget

Den første beslutning er, hvor arbejdet foregår. W3C's arkitekturdokument beskriver et spektrum med en klient, der gør alt, i den ene ende og en server, der gør alt, i den anden, og konstaterer, at de fleste løsninger kombinerer de to. Hvor en applikation hører hjemme, afhænger ifølge dokumentet af klientens kraft, serverens kraft og belastning og båndbredden mellem dem. Det nævner afvejningen fra begge sider: indhold, der skræddersys på serveren, gør caching mindre effektiv, og når antallet af mulige klienter er ubegrænset, skalerer det bedre at skubbe beregningerne ud til dem. De almindelige positioner er sider, der bygges én gang, sider, der renderes på en server pr. forespørgsel, sider, der samles i browseren, og blandinger af dem. En side, der er ens for alle, er billigst at bygge én gang og cache. En side med én kundes data renderes pr. forespørgsel et sted, og vælger man browseren, henter hver læser den kode, der gør arbejdet.

Den anden beslutning er, hvor meget man stoler på den, og det praktiske svar er, at serveren stoler på intet af det. Alt, hvad en frontend leverer, kan læses, ændres og sendes igen af den person, der sidder ved browseren. En skjult knap eller en formular, der nægter at blive sendt, er en høflighed over for en ærlig bruger og en lille omvej for alle andre, så hver tilladelse og validering, der betyder noget, gentages på serveren. Hemmeligheder følger samme regel: en API-nøgle, der når frem til bundlet, er offentlig, uanset hvad variablen hedder. Content Security Policy begrænser, hvad et indsprøjtet script kan gøre, og specifikationen præsenterer den som dybdeforsvar, aldrig som en erstatning for omhyggelig inputvalidering og outputkodning.

Frontenden er også stedet, hvor tilgængelighed vindes eller tabes. WCAG 2.2, W3C-anbefalingen fra december 2024, opstiller testbare succeskriterier, og mange af dem afhænger af valg i frontenden: om en kontrol kan nås og betjenes fra et tastatur, og om den giver hjælpemidler et navn og en rolle. Native HTML-elementer opfylder flere af dem fra starten, og en hjemmelavet widget skal genopbygge hvert af dem i hånden.

Til sidst skal en frontend, der ved, hvordan databasen gemmer ting, ændres, hver gang databasen gør, mens en frontend, der holder sig til præsentation, lader hver side bevæge sig i sit eget tempo.

Sådan ser det ud i praksis

En frontend i Next.js 16 bruger den rette renderingstilstand for hver del af appen. Marketing- og offentlige sider genereres statisk, og de interaktive dele renderes i browseren. På det autentificerede dashboard og de store oversigtslister, hvor N+1-mønstret gjorde én side til en kaskade af rundture, henter en factory ved navn `createPrefetchedServerPage` sidens data på serveren og overlader dem til klienten allerede udfyldt, og en strategi for cache-invalidering er dokumenteret for hver forespørgsel. En side, der er flyttet over på denne factory, kommer op på én rundtur, og den næste side, der flyttes over, arver den adfærd. Siden beskriver arbejdet som en igangværende migrering, hvor resten af den autentificerede app stadig er client-renderet og venter på sin tur bagved.

Grundlæggerens arkitekturbidrag til NextMariner, en maritim platform, giver frontenden én opgave og intet andet. Next.js-frontenden står for præsentation og interaktivitet. Et bevidst tyndt Go-API, bygget med Huma oven på Fiber, router og validerer forespørgsler, anvender sikkerhedsfiltreringen og orkestrerer, mens forretningslogikken bor i PostgreSQL-funktioner. Designet placerer en ændring af en forretningsregel i ét lag, i SQL, frem for at kræve den samme regel i alle tre. Grænserne blev skrevet ned og derefter håndhævet i code review. Det dokumenterede resultat er, at præsentation, håndtering af forespørgsler og forretningsregler har hver sit ansvar.

Arbejdet med at hærde en applikation tager udgangspunkt i, at klienten er fjendtlig. I frontenden sætter Next.js' middleware i `proxy.ts` en Content Security Policy med en nonce pr. forespørgsel og `strict-dynamic`, sammen med HSTS og SameSite-cookies. Selve grænsen ligger på serveren. Rettighederne, det vil sige tier, rolle, organisation og afgrænsning til bestemte skibe, genkontrolleres på serveren ved hver forespørgsel, og frontendens gates behandles kun som UX: de afgør, hvad en person ser, og serveren afgør, hvad personen må.

Hvor det går galt

Fejl i en frontend har som regel samme årsag: noget blev afgjort i browseren, som hørte hjemme på serveren, eller omvendt.

En API-nøgle dukker op i den JavaScript-kode, enhver besøgende kan hente. I Next.js når en miljøvariabel kun frem til browserens bundle, når navnet begynder med `NEXT_PUBLIC_`, og enhver anden variabel, der læses fra klientkode, erstattes med en tom streng. En forespørgsel, der kommer tilbage som uautoriseret, efterfulgt af en hurtig omdøbning for at få den til at virke, lægger hemmeligheden ind i en offentlig fil. Hold kode, der har brug for en hemmelighed, på serveren, og markér dens modul med pakken `server-only`, så en import fra en Client Component får buildet til at fejle. Rotér enhver nøgle, der allerede er udgivet.

```
import "server-only";
export async function getInvoices(customerId: string) {
    const url = `${process.env.BILLING_URL}/customers/${customerId}/invoices`;
    const headers = { Authorization: `Bearer ${process.env.BILLING_KEY}` };
    const res = await fetch(url, { headers });
    return res.json();
}
```

En bruger når frem til poster, som brugerfladen havde skjult. Tilladelsestjekket lå i frontenden som en route guard, et skjult menupunkt eller en deaktiveret knap. Enhver, der læser netværkstrafikken, kan kalde API'et direkte med andre parametre. Tjek hver rettighed på serveren ved hver forespørgsel, mod den indloggede bruger og den konkrete post, og behandl frontendens tjek som en bekvemmelighed for ærlige brugere.

Siden fyldes ud én spinner ad gangen. Hver komponent henter sine egne data, efter at den er renderet, så et barns forespørgsel venter på forælderens, og siden ankommer som et vandfald af rundture. Hent en sides data samlet og tæt på kilden, på serveren hvor frameworket tillader det, og send dem nedad.

Brugere med tastatur og skærmlæser sidder fast i en hjemmelavet kontrol. En klikbar `div` får en knaps museadfærd og intet af resten: tabulatortasten springer den over, Enter og Mellemrum gør intet, og hjælpemidler får intet at vide om, hvad den er. WCAG 2.2 kræver på niveau A, at al funktionalitet virker fra et tastatur (succeskriterium 2.1.1), og at hver komponent oplyser sit navn og sin rolle (4.1.2). Brug `button`, `a`, `input` og `label` til det, de er, og prøv hver ny skærm med tastaturet alene, før den udgives.

Læs mere

- Architecture of the World Wide Web, Volume One — W3C-anbefalingen om user agents og om at fordele arbejdet mellem klient og server.
- HTML Standard — WHATWG's levende standard for de elementer og browser-API'er, en webfrontend bygges af.
- Web Content Accessibility Guidelines (WCAG) 2.2 — W3C-anbefalingen med de testbare succeskriterier bag en tilgængelig brugerflade.
- Content Security Policy Level 3 — W3C-arbejdsudkastet, der definerer nonces og `strict-dynamic`.
- Next.js: Server and Client Components — hvordan ét framework fordeler en frontends kode mellem server og browser.

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 frontend?« 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/frontend/