Engineer

Frontend

the part of a system handling presentation and interaction

What it is

A frontend is the part of a software system that people look at and operate: the pages, screens, forms and controls, along with the code that draws them and responds to input. It is also written front end or front-end, and on the web the same territory is often called the client side.

On the web, a frontend is assembled from three languages a browser understands. HTML gives a page its structure and meaning, CSS decides how it looks, and JavaScript adds behaviour. The browser is what the W3C's Architecture of the World Wide Web calls a user agent, software acting on behalf of a person, which retrieves data from servers and presents it. Everything a frontend sends to it ends up on a device that belongs to someone else.

Its partner is the backend, which stores data, applies the business rules and answers requests. The two talk over HTTP, usually through an API. It is tempting to draw the line between them where the network is, but current frameworks blur it. In Next.js, pages and layouts are Server Components by default: they produce what the user sees and execute on a server. A `"use client"` directive at the top of a file marks the boundary, and that file, its imports and the components it renders join the bundle sent to the browser. What makes code frontend is its job, presentation and interaction, and where it executes is a separate decision.

Use the concept in a decision

Put "What is a frontend?" to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

The first decision is where the work happens. The W3C architecture document describes a spectrum with the client doing everything at one end and the server doing everything at the other, and observes that most designs combine the two. Where an application belongs depends, in the document's terms, on the power of the client, the power and load of the server, and the bandwidth between them. It names the trade-off from both sides: tailoring content on the server lowers caching efficiency, and when the number of possible clients is unbounded, pushing computation out to them scales better. The usual positions are pages built once, pages rendered on a server per request, pages assembled in the browser, and mixtures of these. A page that is the same for everyone is cheapest built once and cached. A page showing one customer's data is rendered per request somewhere, and choosing the browser means every reader downloads the code that does it.

The second decision is how far to trust it, and the working answer is that the server trusts none of it. Everything a frontend delivers can be read, edited and replayed by the person holding the browser. A hidden button or a form that refuses to submit is a courtesy to an honest user and a minor detour for anyone else, so every permission and validation that matters is repeated on the server. Secrets follow the same rule: an API key that reaches the bundle is public, whatever the variable is called. Content Security Policy narrows what an injected script can do, and its specification presents it as defence in depth, never a substitute for validating input and encoding output carefully.

The frontend is also where accessibility is won or lost. WCAG 2.2, the W3C Recommendation published in December 2024, sets testable success criteria, and many of them turn on frontend choices: whether a control can be reached and operated from a keyboard and whether it exposes a name and a role to assistive technology. Native HTML elements meet several of them out of the box, and a custom widget has to rebuild each one by hand.

Finally, a frontend that knows how the database stores things has to change whenever the database does, while one that keeps to presentation lets each side move at its own pace.

What it looks like in practice

Where a frontend renders is decided part by part on one platform, whose Next.js 16 frontend chooses a rendering mode for each part of the app. Marketing and public pages are generated statically, and the interactive parts are rendered in the browser. On the authenticated dashboard and the large directory listings, where the N+1 pattern turned one page into a cascade of round trips, a factory called `createPrefetchedServerPage` fetches a page's data on the server and hands it to the client already filled in, and a cache-invalidation strategy is documented for each data request. A page moved onto the factory loads in one round trip, and the next one to move inherits that behaviour. The migration is still in progress, with the rest of the authenticated app rendered on the client and queued behind it.

The founder's architecture contribution under NextMariner, a maritime platform, gives its frontend one job and nothing else. The Next.js frontend handles presentation and interactivity. A deliberately thin Go API, built with Huma over Fiber, routes and validates requests, applies security filtering and orchestrates, while the business logic lives in PostgreSQL functions. The design puts a business-rule change in one layer, in SQL, instead of requiring the same rule in all three. The boundaries were written down and then enforced in code review. The recorded result is that presentation, request handling and business rules have separate responsibilities.

The work of hardening an application starts from the assumption that the client is hostile. On the frontend, the Next.js proxy middleware in `proxy.ts` sets a Content Security Policy with a nonce generated per request and `strict-dynamic`, along with HSTS and SameSite cookies. The boundary itself sits on the server. Entitlements, covering tier, role, organization and ship scoping, are checked again on the server for every request, and the frontend's gates count as user experience only: they shape what a person sees, and the server settles what that person may do.

Where it goes wrong

Frontend failures tend to share a cause: something was decided in the browser that belonged on the server, or the other way round.

An API key shows up in the JavaScript any visitor can download. In Next.js, an environment variable reaches the browser bundle only when its name starts with `NEXT_PUBLIC_`, and any other variable read from client code is replaced with an empty string. A request that comes back unauthorised, followed by a quick rename to make it work, inlines the secret into a public file. Keep code that needs a secret on the server, and mark its module with the `server-only` package so that importing it from a Client Component fails the build. Rotate any key that has already shipped.

```
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();
}
```

A user reaches records the interface had hidden from them. The permission check lived in the frontend, as a route guard, a hidden menu entry or a disabled button. Anyone who reads the network traffic can call the API directly with other parameters. Check every entitlement on the server for every request, against the signed-in user and the specific record, and treat the frontend's checks as a convenience for honest users.

The page fills in one spinner at a time. Each component fetches its own data after it renders, so a child's request waits for its parent's and the page arrives as a waterfall of round trips. Fetch a page's data together and close to its source, on the server where the framework allows, and pass it down.

Keyboard and screen-reader users get stuck on a custom control. A clickable `div` gets the mouse behaviour of a button and none of the rest: it takes no focus, ignores Enter and Space, and tells assistive technology nothing about what it is. WCAG 2.2 asks at level A that a keyboard can operate all functionality (success criterion 2.1.1) and that every component expose its name and role (4.1.2). Use `button`, `a`, `input` and `label` for what they are, and try each new screen with the keyboard alone before it ships.

Further reading

- Architecture of the World Wide Web, Volume One — the W3C Recommendation on user agents and on splitting work between client and server.
- HTML Standard — the WHATWG living standard for the elements and browser APIs a web frontend is built from.
- Web Content Accessibility Guidelines (WCAG) 2.2 — the W3C Recommendation with the testable success criteria behind an accessible interface.
- Content Security Policy Level 3 — the W3C Working Draft that defines nonces and `strict-dynamic`.
- Next.js: Server and Client Components — how one framework divides a frontend's code between server and browser.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is a frontend?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

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

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/frontend/