Engineer

UI

the part of a program a person sees and works with

What it is

UI is short for user interface: the screens, menus, fields, buttons, messages and sounds through which a person tells a program what to do and finds out what it did. It is also the set of rules behind them, such as which control is reachable from where, what happens on a click and what the program says when something goes wrong.

NIST's glossary takes the widest view, defining a user interface as "the physical or logical means by which users interact with a system, device or process". The physical side is the keyboard, the touch screen, the knob on a device. The logical side is everything drawn or printed in response: a window with menus, a prompt waiting for a typed command, a voice asking for a number. A graphical interface, a command line and a spoken menu are all user interfaces, and one program often has more than one of them.

On the web, WCAG names the building block. A *user interface component* is "a part of the content that is perceived by users as a single control for a distinct function". Its notes count a component by what the user perceives as one control, not by how it was programmed. A single programmatic element can therefore hold several components, and the standard's own example is a control that moves through content by line, by page or to any point, where each way of moving needs its own name and counts as its own component. Form elements, links and controls built by scripts all count.

Two neighbouring terms travel with it. *UX*, user experience, is the wider question of what using the product is like from start to finish: whether the task gets done, how long it takes and how mistakes are handled. The UI is the part of that experience that can be pointed at on a screen. A *design system* is a shared set

of components, tokens and rules that a team builds its UI from, so that two screens built by two people in two different months still look and behave like one product.

Use the concept in a decision

Put “What is a UI?” 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

A program’s value reaches people through its interface or not at all. A feature that can only be found by someone who already knows where it is has, for most users, never shipped. Such a gap rarely shows up in a test suite, because every function works; it shows up in support questions and abandoned forms.

Consistency is learned once and reused. People carry what they learnt on one screen to the next. WCAG 2.2 success criterion 3.2.4, Consistent Identification, asks that components with the same function within a set of web pages be identified consistently. The Understanding document for it gives the reason in plain terms: when the same function is labelled differently in two places, people may assume the two do different things, or spend time working out that they do not.

Errors are part of the interface. Every form meets bad input, and how it answers is as much a design decision as its layout. Success criterion 3.3.1, Error Identification, asks that a detected error name the field and describe the problem in text. Criterion 3.3.3, Error Suggestion, goes further: when the program knows how automatically detected bad input could be corrected, it provides suggestions unless doing so would jeopardize the security or purpose of the content. The Understanding document’s example is a field that expects a month name, where a person who types “12” is offered either the list of accepted values or the question “Do you mean December?”.

Every control has to be explainable to software as well. Criterion 4.1.2, Name, Role, Value, asks that for every user interface component, including those built by scripts, programs can determine its name and role, and that changes in its state are reported to them. A control that only looks like a control on screen is invisible to the assistive technology many people rely on to reach it.

What it looks like in practice

The clearest case is a maritime platform where the product's UI and UX were designed end to end. It shows professionals, companies, ships, jobs and reviews, and had to be dense enough to hold real maritime data without becoming a wall. A handful of decisions carried it. Directory grids switch between cards and a table, and the app remembers which view the person chose. Forms show their requirements above the input as a live list, each rule turning green once it is met and amber while it is not, so people are guided while they type instead of corrected after they submit. Breadcrumbs and a two-column layout repeat on every entity page, which makes separate screens read as one product. Errors lead somewhere: a redirect instead of a dead end. The page makes the trust argument directly: on a platform people use for their careers, a slapdash look would make them trust the data less, and they would be right to.

The commercial side is clearest in 200 UI and UX improvements for a GIS map application. The interface had become powerful and complicated, with good functionality behind clumsy interactions. Rather than guessing, the work started from feedback and usage data, produced a backlog of 200 concrete changes, ranked them by impact and took the ones that mattered most through the team until they shipped as real features. The interface became noticeably easier to use, and the work contributed to a tenfold rise in revenue.

Keeping an interface consistent over time is a separate problem, addressed in an iOS 26 "Liquid Glass" design system. The visual language uses translucent panels with backdrop blur, layered shadows, a touch of refraction and spring animations, built from Tailwind utilities. Every screen composes from a fixed inventory of components, among them Card, Label, Button, GlassIconButton and DirectoryGrid. Lint rules reject a hand-rolled badge or chip and point at the canonical component, and the documentation pairs concrete mistakes with their resolutions. Consistency stops depending on whoever reviews the change that day and is held by the tooling instead.

Where it goes wrong

- **Colour carries the meaning on its own.** A field outlined green for valid and red for invalid tells nothing to someone who cannot tell those colours apart. The Understanding document for Use of Color names exactly this case and asks for a second visual sign. A word, an icon or a change of shape next to the colour does the job.
- **Something drawn to look like a control is not one.** A styled box with a click handler looks like a button to a mouse user and is absent for anyone using a keyboard or a screen reader. The platform's own controls arrive with a role, focus and keyboard behaviour that a lookalike has to rebuild by hand, piece by piece.
- **Every screen reinvents its components.** Each new view gets its own buttons, badges and cards, each slightly different, until a single change means editing five versions of the same thing. One inventory of components, enforced by a linter rather than by memory, keeps the drift out.
- **Errors arrive late and far from their cause.** A form that accepts everything, then returns a red block at the top listing what was wrong, makes people hunt for the fields. Naming the field, describing the problem next to it in words and suggesting a correction turns the same check into guidance.
- **The interface is built for its builder's desk.** A large monitor, a mouse, sharp eyesight and a fast connection describe the person who made it, not the people who will use it. A phone held in one hand, a zoomed page or a keyboard with no mouse each reveal problems that a desk never shows.

Further reading

- NIST CSRC Glossary: user interface — the one-line definition from NIST SP 800-152, covering both the physical and the logical means of interacting with a system.
- WCAG 2.2: user interface component — the definition by perceived control, with notes on how components relate to the elements that implement them.
- Understanding SC 3.2.4: Consistent Identification — why components with the same function should carry the same name across a site, and which repeated components it matters for most.
- Understanding SC 3.3.3: Error Suggestion — what a form should say when it knows how bad input could be fixed, with worked examples.
- Understanding SC 1.4.1: Use of Color — when colour may carry meaning and when a second visual sign is required.

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 UI?” 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/ui/