Engineer

User

a person, or a process acting for one, allowed into a system

What it is

A user is whoever, or whatever, a system acts for. In ordinary technical writing the word does three different jobs, and most confusion about users comes from letting those jobs blur into one.

The first sense is the person: someone who opens a product and tries to get something done. Designers, support staff and product managers usually mean this one, and it has no fixed shape inside the system at all. The second is the account: the record a system keeps so that it can recognise someone again, usually a name to sign in with, an internal identifier, a set of permissions and a status such as active or suspended. Most application code means this one. The third is the operating-system user: on Unix-like systems, an entry in the user database with a user name and a numeric user ID. Running processes carry user-ID credentials that the kernel uses for permission checks.

The three rarely line up one to one. A person can hold accounts in many services, and NIST's digital identity guidelines say plainly that a person may have several digital identities, each unique only inside its own service. One account can be shared by several people, which is nearly always a mistake. And a great many users have no person behind them at all: the web server, the database, the backup job and the deploy script each run as an operating-system user of their own, and often hold accounts in other systems as well. That is why the NIST definition reads "individual, or (system) process acting on behalf of an individual".

On a Unix host the split between name and number is easy to see:

```
$ id ana
uid=1001(ana) gid=1001(ana) groups=1001(ana)
```

The kernel works with the number. POSIX defines a user database holding, for each user, the user name, the numerical user ID, an initial group ID, an initial working directory and an initial program to run. File ownership is stored as the number, and the name is looked up only when something is shown to a person. A process also carries more than one user ID: a real one, which says who started it, and an effective one, which says whose permissions it is using at that moment.

Application accounts usually make the same separation. The SCIM standard, used to provision users from one system into another, requires a `userName` that people see and sign in with, unique within the service and compared without regard to case, and a separate `id` that the service assigns, that stays stable and is never reassigned. An `active` flag tells a suspended account apart from a working one.

Use the concept in a decision

Put “What is a user?” to work in a real decision. We can help relate the concept to the system you are building or buying. The laptop, the printer, the login and the mailbox, sorted quickly and in plain language, without anyone feeling foolish.

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. Established a Technical Support department — 10,000 clients served by a support department built from scratch

Established a Technical Support department Get in touch →

Why it matters

Every access decision is a decision about a user, so a muddled idea of who the user is turns into a muddled decision. The question “may this person see this invoice?” is answered in code as “does this account, in this context, hold a permission that covers this record?”, and each step of that translation is a place where the intended person and the actual account can drift apart.

The line between person and account is also where accountability lives. Logs record accounts. When each account belongs to exactly one person, a log line says who did something. When an account is shared, the log says which secret was used and nothing about who typed it, and ending one person’s access means changing it for everybody.

Operating-system users are a long-established tool for least privilege. A service that runs as its own unprivileged user is limited by that user’s permissions, not just by file ownership. It can still modify shared writable resources, make network

requests and use credentials available to it. Separate service users reduce the damage a compromise can do; they are not complete isolation. The same service running as the superuser can put the whole machine at risk. Giving each program a user of its own is a small configuration change that narrows the permissions available when something goes wrong.

Finally, people and accounts have different lifetimes. People change their names, their email addresses and their employers; accounts are suspended, merged and deleted. A system that treats a changeable attribute, such as an email address, as the identity of an account will one day attach one person's history to someone else.

What it looks like in practice

On a maritime platform, professionals manage organisations, such as companies and academies, that have no logins of their own. The page states the rule directly: the person is the account, and the organisation is something they act on behalf of. The work was a context switch that lets a user act as any organisation they manage. The browser's local storage holds which organisation is active, so switching is instant, and a server-side cookie mirrors it so that pages rendered on the server show the same context. None of that decides access. Authorisation is checked again on the server for every request, so someone who edits the stored value changes only what their own screen shows.

On the company's own host, releases are shipped by an operating-system user with almost no rights. When the container plane was set up, deployment was separated from provisioning: root prepares the deployment path once, and after that an unprivileged user redeploys by running one fixed script through a scoped rule that permits that command and no arguments. A routine release builds the image and restarts the service, and nobody opens a root shell to do it.

Knowing who the users are is itself a report. One of seven read-only reporting roles renders an access report from the live host into Markdown in the repository, with accounts, sudo scope, key fingerprints, users of the git forge and database roles side by side. The seven reports share one rule: a figure appears only when the run measured it, and a gap stays blank rather than taking a plausible guess that a reader would believe. Taken together, the reports have found real things, among them an undeclared listening port and two dead security controls.

Where it goes wrong

A deleted user comes back as someone else. An account is removed, its identifier is later issued again, and the newcomer inherits whatever still points at the old value: rows in a table, a place on a list, files on disk. On a Unix host the files a removed user owned keep their numeric owner, and the next user created with that number owns them. Give accounts identifiers that are never reused, as SCIM requires of its `id`, and decide what happens to a departing user's data before removing the user.

An email address is used as the account. It looks like a natural key until someone changes address, loses access to an old one, or an employer passes an address on to a new member of staff. Then one person's history is attached to another, or one person ends up with two accounts. Keep a stable internal identifier as the key and treat the address as an attribute that can change.

Everyone signs in as the same account. A shared administrator login is quick to set up and impossible to audit. When someone leaves, their access ends only if the secret changes for everyone, and the log names a role instead of a person. One account per person, with shared power granted through groups, roles or a narrowly scoped sudo rule, keeps both the log and the departures honest.

The browser decides what the user may do. A role kept in local storage, a cookie or a hidden form field is a suggestion from the client, and anyone can edit it. If the server trusts it, editing it is an attack. The client may remember context for convenience; the server has to check permissions itself, on every request.

A service runs as root because it was simpler. Everything works, but an exploitable flaw in that service can put the whole host at risk. Give each service its own unprivileged user, and grant any extra right narrowly, for one named command, rather than broadly.

Further reading

- NIST CSRC glossary: user — the definitions used across NIST security publications, each with the document it comes from.
- POSIX.1-2024 Base Definitions: Definitions — the formal meaning of user database, user ID and user name on Unix-like systems.
- RFC 7643: SCIM Core Schema — the User resource for moving accounts between services, with `userName`, `id` and `active`.

- NIST SP 800-63-4: Digital Identity Guidelines — how one person can hold several digital identities, and the roles of applicant, subscriber and claimant.
- POSIX.1-2024: File Access Permissions — access through owner, group and other permission classes, alternate access controls and appropriate privileges; ownership alone does not bound access.

Glossary

Choose for your actual constraints

Describe the device or service, the observed symptoms and the data that must be preserved. We can establish the safest useful diagnostic step.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a user?” 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/user/