Engineer

URL

the address that says where a thing is

What it is

A URL, short for Uniform Resource Locator and called a web address or simply a link in everyday speech, is a line of text that both names a resource and tells software how to fetch it: which protocol to speak, which machine to ask and what to ask that machine for. Paste one into a browser, a terminal or a config file and each of them can act on it without further help.

Its anatomy is fixed by the generic syntax every scheme shares:

```
https://shop.example.com:8443/orders/42?view=full#items

scheme     https
host       shop.example.com
port       8443
path       /orders/42
query      view=full
fragment   items
```

The scheme says how to reach the thing, here HTTP over TLS. The host and optional port say where, the path says which resource on that host, and the query carries parameters, usually as `name=value` pairs joined by `&`. The fragment after `#` points inside the result, at a heading or a position. It belongs to the client: the browser leaves it out of the HTTP request target. That does not make it secret from the server, because client-side code can read it and explicitly send it in another request.

Because characters such as `/`, `?`, `#`, `&` and `=` have jobs in that structure, data that would conflict with a delimiter in the component being built has to be percent-encoded: written as `%` followed by two hexadecimal digits. Which

characters need that treatment depends on the component and its syntax. A space becomes `%20`, an ampersand inside a query parameter value `%26`, and the percent sign itself `%25`. Letters outside ASCII travel the same way, as their UTF-8 bytes, which is how Danish or Ukrainian words in a path get through.

Is a URL the same as a URI? Nearly. A URI, a Uniform Resource Identifier, is any string in that syntax that identifies something; a URL is the kind that also says how to locate it, while `urn:isbn:9780000000002` names a book and gives no way to fetch it. RFC 3986 asks new specifications to say URI. The WHATWG URL Standard, which browsers follow, went the other way: it settled on URL as the one word and set out to describe what real implementations do, aiming to replace the RFC. The two agree on ordinary addresses and differ on edge cases, which is worth remembering when a browser and a server library parse the same odd string.

Use the concept in a decision

Put “What is a URL?” to work in a real decision. We can help relate the concept to the system you are building or buying. A site that is quick, findable and editable by your own staff, without a bill for every change of wording.

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. Developed 100 web applications using HTML/HTML5 — 100 web applications built

Developed 100 web applications using HTML/HTML5 Get in touch →

Why it matters

A published URL is a promise. It ends up in bookmarks, search indexes, emails, printed brochures, chat logs and other people’s pages, and none of those copies update themselves. Changing the address later breaks every one of them unless the old address keeps answering with a redirect to the new one.

That makes the useful decision an early one: what the address is built from. Anything that is likely to change is a poor foundation, whether a page title, a category tree, a product name or a file extension that reveals the current framework. A short identifier, or a slug stored as data and edited on purpose, lasts. So does a plan for what happens when an address has to move anyway: the old one becomes a permanent redirect rather than a 404.

The second decision is what stays out. RFC 9110 puts it bluntly: addresses are made to be shared, not secured. They are written to server and proxy logs, kept in browser history, copied into analytics and, depending on policy, sent along in the `Referer` header to whatever the page loads next. Session identifiers, reset tokens, API keys and personal data belong in headers or request bodies.

Length is a smaller question. The HTTP specification recommends that software accept addresses of at least 8,000 octets, which is generous, but a query string carrying a whole form or a serialized filter tends to hit a shorter limit somewhere along the chain before it reaches that.

What it looks like in practice

The work on this site treats addresses as data that outlives the text on the page. When 495 achievement pages across three languages were generated from a read-only SQLite export at the documented August 2026 implementation stage, the most consequential decision concerned the address. The site had derived a page's URL from the leading words of its English statement, so correcting a sentence silently moved the page and broke every link to it. The address was then authored as data, one row per address per achievement: the first row was the current one, and every later row was a retired address the site emitted as a redirect. The recorded result was that every address published by that stage still answered.

A note on mobile wallets shows what happens when one URL travels inside another. Phantom, Solflare and Backpack open a dApp through universal links that carry the target address in their own path and the requesting origin in a `ref` parameter, each passed through `encodeURIComponent`. An unencoded `?` or `&` in the target corrupts the parse, and the wallet opens on its home screen instead of the browser tab. A guessed custom scheme such as `backpack://` fares worse: the app registers no route that matches it, and the user lands on the install page.

Addresses also serve as identity. When 963 structured-data blocks were replaced by one linked graph, the identifiers for the organisation, the site and the person were minted from the site's origin, and list pages publish their items by URL alone. At the documented July 2026 implementation stage, that choice was measured: naming the items on the portfolio index would have meant 107 full statements and a 52 per cent larger compressed page. The check guarding the graph asserts that every URL it names was actually built.

Where it goes wrong

URL bugs tend to be quiet. The address still looks right; it simply points somewhere else, or reveals more than intended.

Encoding happens zero times or twice. A return address dropped raw into a query, as in `?next=/cart?item=4&size=m`, splits at the unencoded `&`, and the server sees `size=m` as its own parameter. Encoding an already encoded string turns `%20` into `%2520`, and the page receives a literal `%20`. Encode each component exactly once, at the moment it is placed into the address, with the function meant for components. In JavaScript that is `encodeURIComponent`; `encodeURI` leaves `?`, `&` and `#` alone because it expects a whole address.

A secret rides in the query string. A password-reset token or an API key in the URL is copied into access logs, browser history and analytics, and under a permissive referrer policy it is sent in `Referer` to every third party the page loads from. Put credentials in a header. When a token has to travel in a link, as in a reset email, make it single-use and short-lived, and set a `Referrer-Policy` that keeps it at home.

One page answers at five addresses. `http` and `https`, with and without `www`, with and without a trailing slash, `/About` and `/about`. The scheme and host are case-insensitive by definition; the path is case-sensitive unless the server decides otherwise. Search engines split the page's standing across the copies, caches store each one separately and analytics count them apart. Pick one form, redirect every other form to it permanently, and name it as canonical.

A relative link resolves against the wrong base. A reference such as `img/logo.png` replaces everything after the last slash of the current path. On `/docs/guide` it resolves to `/docs/img/logo.png`; on `/docs/guide/` it resolves to `/docs/guide/img/logo.png`. One trailing slash moves every relative link on the page. Root-relative links, starting with `/`, sidestep the problem.

Logic depends on the fragment. Server-side redirects, access logs and analytics that read the HTTP request target do not receive `#` or its fragment. A client can send it separately when the application needs it. Anything that has to reach the server as part of the initial request belongs in the path or the query.

Further reading

- RFC 3986: Uniform Resource Identifier (URI): Generic Syntax — the grammar, percent-encoding, relative resolution and normalisation.
- WHATWG URL Standard — the parsing algorithm browsers implement, step by step.
- RFC 3987: Internationalized Resource Identifiers — how addresses written in any script map onto percent-encoded bytes.
- RFC 8820: URI Design and Ownership — why the owner of a host, and nobody else, decides how its paths look.
- RFC 9110, section 17.9 — the security note on sensitive information in addresses.

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 URL?” 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/url/