Engineer

API

the described way one program calls another

What it is

An API, short for application programming interface, is the published set of operations a piece of software offers to other software: what each operation is called, what it accepts, what it returns and how it reports trouble. The plural is APIs. The caller learns that surface and nothing behind it, so the program on the far side can be rewritten, moved to another machine or split in three, and the caller keeps working as long as the surface holds.

The word covers several layers. A library's API is the functions and types it exports, called inside one process. An operating system's API is its set of system calls, and POSIX standardises a common core of them so that a program written against `open` and `read` builds on more than one Unix-like kernel. A web API is offered across a network, almost always over HTTP, and that is what most people mean by the bare word today: a server answering requests at addresses such as `/v1/orders`, usually with JSON in the body.

Web APIs come in a handful of styles. Resource-oriented ones, often called REST-style, give each thing an address and use HTTP's methods to read and change it. RPC-style ones, such as gRPC, publish named procedures with their arguments and results declared in advance. GraphQL offers one endpoint and a type system, and the client writes a query naming exactly the fields it wants back. Webhooks turn the direction round: the provider calls an address the customer registered when something happens.

Three neighbouring words get mixed up with it. A protocol, such as HTTP, defines how messages travel; an API is what one particular service says in that protocol. An SDK is a library in one programming language that wraps someone's API, so a developer calls a function instead of assembling requests. And the implementation is the code behind the API, which a caller never needs to read.

Use the concept in a decision

Put “What is an API?” to work in a real decision. We can help relate the concept to the system you are building or buying. The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

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. Designed and ran 6 ETL/ELT pipelines — 6 ETL pipelines across three database engines

Designed and ran 6 ETL/ELT pipelines Get in touch →

Why it matters

An API is the part of a system that other people build on, which makes it a promise that outlives the code behind it. Rewriting a handler takes an afternoon. Changing what that handler returns can break an app already installed on customers’ phones, a partner’s nightly import or a script nobody remembers writing.

So the first decision is about audience. A private API, used only by the team’s own frontend, can change in the same pull request as its one client. A public API changes through deprecation and notice, and every field it ships is a field to support for as long as anyone reads it.

The second decision is how strict the contract is. A strict API declares a type and a limit for every input, rejects what it does not recognise and answers each failure with a precise status code. It catches mistakes at the door and is easier to document, test and generate clients for. A lenient API accepts whatever arrives and does its best, which feels friendly in the first month and then turns every accident of the implementation into behaviour somebody depends on. Strictness costs design time up front; leniency costs more later, at a moment nobody chooses.

The third decision concerns other people’s APIs. A payment provider, a mapping service or a language model is an API the product calls and does not control: prices move, versions are retired, quotas tighten. Calling a vendor directly from fifty places makes a switch a rewrite. Calling it from one module the team owns makes the switch a change to that module. The wrapper costs a little indirection and buys a way out.

Semantic Versioning gives this promise a vocabulary: the major number goes up when the public API changes in a way that is incompatible, so a client can tell from the version alone whether an upgrade is safe. Over HTTP there are several places to signal a break, and RFC 9205, the IETF's guidance for protocols built on HTTP, lists a new link relation, a new media type or a new header field among them.

What it looks like in practice

The APIs behind several projects on this site were shaped by one question: what belongs in the API, and what belongs somewhere else.

On NextMariner, a maritime platform for professional networking, company reviews and sponsored education, the founder's architecture contribution included keeping the Go API deliberately thin. Built with Huma over Fiber, it routes, validates the request, applies the security filtering and orchestrates, and holds no business logic at all. The rules live in PostgreSQL functions, so a rule change happens in one layer, in SQL. The boundaries were written down and enforced in review. That makes the API a boring adapter, in the good sense.

A related project took the shaping out of the API as well. PostgreSQL functions assemble each response as JSON, and the Go handler receives it as `json.RawMessage` and forwards it untouched, through a small `QueryJSON` helper. The DTOs, mappers and response structs a layered API usually collects fell away, handlers got dramatically shorter, and the frontend no longer depended on Go. The trade-off is worth naming: the shape of the API's responses is now defined in SQL, so that is where a reviewer reads the contract.

An API that anyone can reach also has to defend itself. A public directory of companies and professionals sits behind 22 rate limiters, each built for the path it protects, with their state in Redis so a limit holds across instances. When Redis is unreachable the limiters fail closed and refuse traffic. Cloudflare Turnstile, an origin lock, a bespoke challenge on the directory and 351 lines of idempotency middleware, which stops a retried write landing twice, complete the defences. The page states the cost plainly: a Redis problem becomes a problem users can see, which was preferred to a Redis problem becoming a billing one.

Where it goes wrong

API failures tend to arrive late, once the clients and the data have grown.

A renamed field breaks a client nobody knew about. Once an API is published, someone depends on every observable detail: field names, the order of a list, the wording of an error. Rename `customer_name` to `name` and a partner's integration breaks on the next request. Change by addition instead: add the new field beside the old one, mark the old one deprecated, watch the logs until nobody reads it, and remove it only in a new major version.

A list endpoint gets slower every month. It returns every matching row, which was fine with fifty records at launch and times out at two hundred thousand. Paginate from the first release, with a default page size and a hard maximum the client cannot exceed. For large or busy collections prefer a cursor to an offset: with an offset, a row inserted between two requests pushes another row onto the next page, where the client reads it twice.

```
GET /v1/orders?limit=50&after=ord_8f2c HTTP/1.1
Host: api.example.com
```

One customer reads another customer's order. The API checks that the caller is signed in and then loads `/v1/orders/42` by its identifier alone, so changing the number in the address walks through other people's data. Authentication answered who is calling; nothing asked whether that caller owns order 42. Scope every lookup by owner inside the query itself, and keep a test that signs in as one account and requests another account's records.

The response carries columns nobody meant to publish. An endpoint serialises a database row or an ORM object as it stands, so a password hash, an internal flag or a supplier's cost price goes out to every client, and a column added next year is published the day it is created. The same mistake on input lets a request set `role` or `is_admin` by including it in the body. Give each endpoint explicit input and output types that list the fields allowed, so publishing a field becomes a decision someone makes.

Further reading

- RFC 9110: HTTP Semantics — the methods, status codes and headers that every web API is built from, defined by the IETF.
- RFC 9205: Building Protocols with HTTP — best current practice for designing an application on top of HTTP, including how to evolve it.

- RFC 9457: Problem Details for HTTP APIs — a standard JSON shape for error responses, so clients parse every failure the same way.
- RFC 6585: Additional HTTP Status Codes — where 429 Too Many Requests and its `Retry-After` hint for rate-limited clients are defined.
- Semantic Versioning 2.0.0 — the rules for signalling an incompatible API change through a version number.

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 an API?” 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/api/