

CORS (Cross-Origin Resource Sharing)

[CORS or "Cross-Origin Resource Sharing"](#) refers to the situations when a frontend running in a browser has JavaScript code that communicates with a backend, and the backend is in a different "origin" than the frontend.

Origin

An origin is the combination of protocol (`http` , `https`), domain (`myapp.com` , `localhost` , `localhost.tiangolo.com`), and port (`80` , `443` , `8080`).

So, all these are different origins:

- `http://localhost`
- `https://localhost`
- `http://localhost:8080`

Even if they are all in `localhost` , they use different protocols or ports, so, they are different "origins".

Steps

So, let's say you have a frontend running in your browser at `http://localhost:8080` , and its JavaScript is trying to communicate with a backend running at `http://localhost` (because we don't specify a port, the browser will assume the default port `80`).

Then, the browser will send an HTTP `OPTIONS` request to the backend, and if the backend sends the appropriate headers authorizing the communication from this different origin (`http://localhost:8080`) then the browser will let the JavaScript in the frontend send its request to the backend.

To achieve this, the backend must have a list of "allowed origins".

In this case, it would have to include `http://localhost:8080` for the frontend to work correctly.

Wildcards

It's also possible to declare the list as `"*"` (a "wildcard") to say that all are allowed.

But that will only allow certain types of communication, excluding everything that involves credentials: Cookies, Authorization headers like those used with Bearer Tokens, etc.

So, for everything to work correctly, it's better to specify explicitly the allowed origins.

Use `CORSMiddleware`

You can configure it in your **FastAPI** application using the `CORSMiddleware` .

- Import `CORSMiddleware` .
- Create a list of allowed origins (as strings).
- Add it as a "middleware" to your **FastAPI** application.

You can also specify if your backend allows:

- Credentials (Authorization headers, Cookies, etc).
- Specific HTTP methods (`POST` , `PUT`) or all of them with the wildcard `"*"` .

- Specific HTTP headers or all of them with the wildcard `"*"` .

```
{!../../../docs_src/cors/tutorial001.py!}
```

The default parameters used by the `CORSMiddleware` implementation are restrictive by default, so you'll need to explicitly enable particular origins, methods, or headers, in order for browsers to be permitted to use them in a Cross-Domain context.

The following arguments are supported:

- `allow_origins` - A list of origins that should be permitted to make cross-origin requests. E.g. `['https://example.org', 'https://www.example.org']` . You can use `['*']` to allow any origin.
- `allow_origin_regex` - A regex string to match against origins that should be permitted to make cross-origin requests. e.g. `'https://.*\.example\.org'` .
- `allow_methods` - A list of HTTP methods that should be allowed for cross-origin requests. Defaults to `['GET']` . You can use `['*']` to allow all standard methods.
- `allow_headers` - A list of HTTP request headers that should be supported for cross-origin requests. Defaults to `[]` . You can use `['*']` to allow all headers. The `Accept` , `Accept-Language` , `Content-Language` and `Content-Type` headers are always allowed for CORS requests.
- `allow_credentials` - Indicate that cookies should be supported for cross-origin requests. Defaults to `False` . Also, `allow_origins` cannot be set to `['*']` for credentials to be allowed, origins must be specified.
- `expose_headers` - Indicate any response headers that should be made accessible to the browser. Defaults to `[]` .
- `max_age` - Sets a maximum time in seconds for browsers to cache CORS responses. Defaults to `600` .

The middleware responds to two particular types of HTTP request...

CORS preflight requests

These are any `OPTIONS` request with `Origin` and `Access-Control-Request-Method` headers.

In this case the middleware will intercept the incoming request and respond with appropriate CORS headers, and either a `200` or `400` response for informational purposes.

Simple requests

Any request with an `Origin` header. In this case the middleware will pass the request through as normal, but will include appropriate CORS headers on the response.

More info

For more info about CORS, check the [Mozilla CORS documentation](#).

!!! note "Technical Details" You could also use `from starlette.middleware.cors import CORSMiddleware` .

```
**FastAPI** provides several middlewares in `fastapi.middleware` just as a convenience for you, the developer. But most of the available middlewares come directly from Starlette.
```