**Fetch: Cross-Origin Requests**

 Cross-Origin Requests is a security feature implemented by web browsers to control and restrict web pages from making requests to a different domain than the one from which the web page originated.  
  
This security measure is designed to prevent malicious websites from accessing sensitive data on other websites without permission.  
  
To successfully make cross-origin requests with fetch():  
try {  
  
await fetch('http://example.com');  
  
} catch(err) {  
  
alert(err); // Failed to fetch  
  
}  
  
Fetch fails, as expected, because cross-origin restrictions protect the internet from evil hackers.

**Using Forms**

 One way to communicate with another server was to submit a <form> there. People submitted it into <iframe>, just to stay on the current page.  
  
it was possible to make a GET/POST request to another site, even without networking methods. But as it’s forbidden to access the content of an <iframe>from another site, it wasn’t possible to read the response.  
  
As we can see, forms allowed to send data anywhere, but not receive the response. To be precise, there were actually tricks for that (required special scripts at both the iframe and the page), but let these dinosaurs rest in peace.

**Simple Requests**

 A simple request is a request that satisfies two conditions:  
  
Simple method: GET, POST or HEAD  
  
Simple headers – the only allowed custom headers are:  
  
Accept,  
Accept-Language,  
Content-Language,  
Content-Type with the value application/x-www-form-urlencoded, multipart/form-data or text/plain.

**CORS for Simple Requests**

 If a request is cross-origin, the browser always adds Origin header to it.  
  
For instance, if we request https://anywhere.com/request from https://javascript.info/page, the headers will be like:  
  
GET /request  
Host: anywhere.com  
Origin: https://javascript.info  
  
As you can see, Origin contains exactly the origin (domain/protocol/port), without a path.  
  
The server can inspect the Origin and, if it agrees to accept such a request, adds a special header Access-Control-Allow-Origin to the response. That header should contain the allowed origin (in our case https://javascript.info), or a star \*. Then the response is successful, otherwise an error.  
  
The browser plays the role of a trusted mediator here:  
  
It ensures that the correct Origin is sent with a cross-domain request.  
  
If checks for correct Access-Control-Allow-Origin in the response, if it is so, then JavaScript access, otherwise forbids with an error.

**Response Headers**

 For cross-origin request, by default JavaScript may only access “simple response headers”:  
  
Cache-Control  
Content-Language  
Content-Type  
Expires  
Last-Modified  
Pragma  
  
Any other response header is forbidden.

**Non-simple Requests**

 Non-simple requests refer to certain types of HTTP requests made from a web page to a different domain (origin) using JavaScript's XMLHttpRequest or the Fetch API.  
These requests are considered non-simple because they trigger additional security checks due to the potential for cross-origin security issues.  
  
A preflight request uses method OPTIONS and has nobody.  
  
Access-Control-Request-Method header has the requested method.  
Access-Control-Request-Headers header provides a comma-separated list of non-simple HTTP-headers.  
If the server agrees to serve the requests, then it should respond with status 200, without body.  
  
The response header Access-Control-Allow-Methods must have the allowed method.  
The response header Access-Control-Allow-Headers must have a list of allowed headers.  
Additionally, the header Access-Control-Max-Age may specify a number of seconds to cache the permissions. So the browser won’t have to send a preflight for subsequent requests that satisfy given permissions.

**Credentials**

 A cross-origin request by default does not bring any credentials (cookies or HTTP authentication).  
  
That’s uncommon for HTTP-requests. Usually, a request to http://site.com is accompanied by all cookies from that domain. But cross-domain requests made by JavaScript methods are an exception.  
  
For example, fetch('http://another.com') does not send any cookies, even those that belong to another.com domain.  
  
Why?  
  
That’s because a request with credentials is much more powerful than an anonymous one. If allowed, it grants JavaScript the full power to act and access sensitive information on behalf of a user.  
  
Does the server really trust pages from Origin that much? Then it must explicitly allow requests with credentials with an additional header.  
  
To send credentials, we need to add the option credentials.

**Fetch API**

 The Fetch API is a modern JavaScript interface for making network requests, such as fetching resources from a server or making API calls, in a web browser.  
It provides a more powerful and flexible alternative to the older XMLHttpRequest (XHR) API.  
The Fetch API is designed to be more straightforward to use and is based on promises, making it easier to work with asynchronous operations.