**Download Your Daily Notes**

**My own views on Fetch: Cross-Origin Requests**

 The core concept here is origin – a domain/port/protocol triplet.  
  
Cross-origin requests – those sent to another domain (even a subdomain) or protocol or port – require special headers from the remote side. That policy is called “CORS”: Cross-Origin Resource Sharing.  
  
Syntax:  
  
try {  
  
await fetch('http://example.com');  
  
} catch(err) {  
  
alert(err); // Failed to fetch  
  
}

**Daily Notes - Fetch: Cross-Origin Requests**

 The core concept here is origin – a domain/port/protocol triplet.  
  
Cross-origin requests – those sent to another domain (even a subdomain) or protocol or port – require special headers from the remote side. That policy is called “CORS”: Cross-Origin Resource Sharing.  
  
Syntax:  
  
try {  
  
await fetch('http://example.com');  
  
} catch(err) {  
  
alert(err); // Failed to fetch  
  
}

**Think and Reflect**

 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.

**Daily Notes - 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

**Daily Notes - Simple Requests**

 There are two types of cross-domain requests:  
1. Simple requests.  
2. All the others.  
  
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.  
  
Any other request is considered “non-simple”. For instance, a request with PUT method or with an API-Key HTTP-header does not fit the limitations.  
  
The essential difference is that a “simple request” can be made with a <form> or a <script>, without any special methods.  
  
Contrary to that, requests with non-standard headers or e.g. method DELETE can’t be created this way. For a long time, JavaScript was unable to do such requests. So an old server may assume that such requests come from a privileged source, “because a webpage is unable to send them”.

**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

**Daily Notes - 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.

**Daily Notes - Non-simple Requests**

 We can use any HTTP-method: not just GET/POST, but also PATCH, DELETE and others.  
To avoid misunderstandings, any “non-simple” request – that couldn’t be done in the old times, the browser does not make such requests right away. Before it sends a preliminary, so-called “preflight” request, asking for permission.  
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.

**Think and Reflect**

 By default, cross-origin requests made by JavaScript methods do not send any credentials (such as cookies or HTTP authentication). This is because allowing such requests grants JavaScript full access to sensitive information on behalf of the user. To send credentials, the option credentials: "include" must be added. The server must also explicitly allow requests with credentials by adding the header Access-Control-Allow-Credentials: true. Cross-origin requests are split into "simple" and non-simple requests. Simple requests can only use certain headers and are sent right away, while non-simple requests require a preliminary "preflight" request. The Origin header is used in cross-origin requests because the Referer header is not always present and can be unreliable.

**Daily Notes - Credentials**

 By default, cross-origin requests made by JavaScript methods do not send any credentials (such as cookies or HTTP authentication). This is because allowing such requests grants JavaScript full access to sensitive information on behalf of the user. To send credentials, the option credentials: "include" must be added. The server must also explicitly allow requests with credentials by adding the header Access-Control-Allow-Credentials: true. Cross-origin requests are split into "simple" and non-simple requests. Simple requests can only use certain headers and are sent right away, while non-simple requests require a preliminary "preflight" request. The Origin header is used in cross-origin requests because the Referer header is not always present and can be unreliable.

**Daily Notes - Fetch API**

 Provides a comprehensive list of all possible fetch options along with their default values. The options include referrer, referrerPolicy, mode, credentials, cache, redirect, and integrity.  
The referrer and referrerPolicy options govern how fetch sets the HTTP Referer header, while mode serves as a safeguard that prevents cross-origin requests. The credentials option specifies whether fetch should send cookies and HTTP-Authorization headers with the request. The cache options allow us to ignore HTTP-cache or fine-tune its usage. The redirect option allows us to change the way fetch handles HTTP-redirects, while integrity allows us to check if the response matches the known-ahead checksum.

**Daily Notes - Activity 1 - Applying new Concepts**

 This has been completed and implemented on the Mzamomtsha Primary School Website.

**My Views on the Day**

 1. Learning about forms, response headers, requests, non-simple requests, credentials and fetch API.  
  
2. Activity 1.  
  
3. None.  
  
4. None.

**Daily Notes - Day 3 Reflections**

 1. Learning about forms, response headers, requests, non-simple requests, credentials and fetch API.  
  
2. Activity 1.  
  
3. None.  
  
4. None.