API Advanced

Cross-Origin Resource Sharing (CORS)

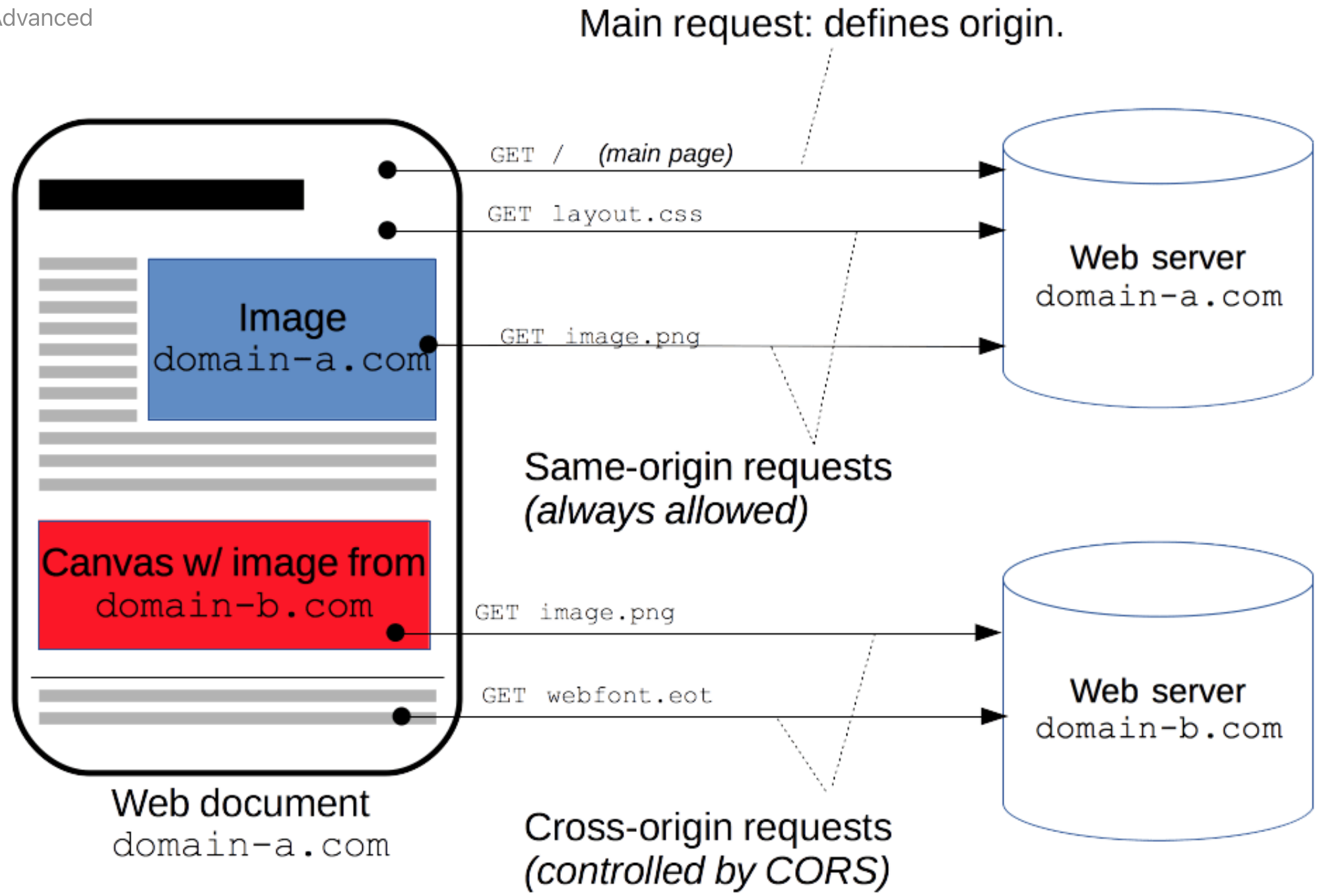
Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.

CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

An example of a cross-origin request: the front-end JavaScript code served from `https://domain-a.com` uses `fetch()` to make a request for `https://domain-b.com/data.json`.

For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts. For example, `fetch()` and `XMLHttpRequest` follow the same-origin policy. This means that a web application using those APIs can only request resources from the same origin the application was loaded from *unless the response from other origins includes the right CORS headers*.

The CORS mechanism supports secure cross-origin requests and data transfers between browsers and servers. Browsers use CORS in APIs such as `fetch()` or `XMLHttpRequest` to mitigate the risks of cross-origin HTTP requests.


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image
domain-a.com
Web server
domain-a.com
Same-origin requests
(always allowed)
Canvas w/ image from
domain-b.com
GET image.png
GET webfont.eot
Web server
domain-b.com
Web document
domain-a.com
Cross-origin requests
(controlled by CORS)

CORS errors

If the CORS configuration isn't set up correctly, the browser console will present an error like `"Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at $somesite"` indicating that the request was blocked due to violating the CORS security rules.

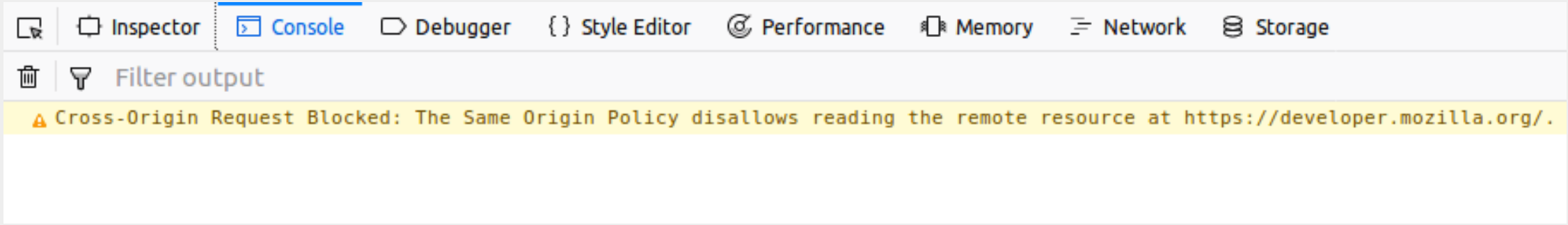
This might not necessarily be a set-up mistake, though. **It's possible that the request is in fact intentionally being disallowed by the user's web application and remote external service.**

However, If the endpoint is meant to be available, some debugging is needed to succeed.

Identifying the issue

To understand the underlying issue with the CORS configuration, you need to find out which request is at fault and why. These steps may help you do so:

1. Navigate to the website or web app in question and open the Developer Tools.
2. Now try to reproduce the failing transaction and check the `console` if you are seeing a CORS violation error message. It will probably look like this:



Firefox's console displays messages in its console when requests fail due to CORS. Part of the error text is a "reason" message that provides added insight into what went wrong. The reason messages are listed here; click the message to open an article explaining the error in more detail and offering possible solutions.

Note: Some times, when it's not a bug there's not a lot you can do...

JWT (JSON Web Tokens)

A **JWT** is a structured security token format used to encode JSON data.

The main reason to use JWT is to exchange JSON data in a way that can be cryptographically verified.

Source Encode or Decode JWTs

Types of JWTs

There are two types of JWTs:

- JSON Web Signature (JWS)
- JSON Web Encryption (JWE)

The data in a **JWS** is *public* — meaning anyone with the token can read the data — whereas a **JWE** is encrypted and *private*.

To read data contained within a JWE, you need both the token and a secret key.

When you use a JWT, it's usually a JWS. The 'S' (the *signature*) is the important part and allows the token to be validated.

How JWTs Are Used

OAuth 2.0 identity providers (IdP) commonly use JWTs for access tokens. You may have seen an HTTP request with an authorization header that looks like this:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJuYW1lIjoiSm9lIENvZGVyIn0.5dlp7GmziL2QS06sZgK4mtaqv0_xX4oFUuTDh1zHK4U
```

> JWT access tokens are NOT part of the OAuth 2.0 specification, but almost all IdPs support them.

Using a JWT (*actually a JWS*) allows the token to be validated locally, without making an HTTP request back to the IdP, thereby increasing your application's performance.

Applications can make use of data inside the token, further reducing expensive HTTP calls and database lookups.

JWT Structure

A JWS contains three parts separated by a dot (.). The first two parts (the "header" and "payload") are Base64-URL encoded JSON, and the third is a cryptographic signature.

Let's look at an example JWT:

```
eyJhbGciOiJIUzI1NiJ9.eyJuYW1lIjoiSm9lIENvZGVyIn0.5dlp7GmziL2QS06sZgK4mtaqv0_xX4oFUuTDh1zHK4U
```

Breaking this down into the individual sections we have:

```
eyJhbGciOiJIUzI1NiJ9 // header: base64 url-decoded {"alg":"HS256"}
.
eyJuYW1lIjoiSm9lIENvZGVyIn0 // payload: base64 url-decoded: {"name":"Joe Coder"}
.
5dlp7GmziL2QS06sZgK4mtaqv0_xX4oFUuTDh1zHK4U /// signature
```

> If you have a JWT with more than three sections, it's probably a JWE.

Getting a token for the Noroff API, Using Postman:

In the Noroff API Collection (or a new collection):

Make a POST request to the `/auth/login` endpoint, with the body:

```
{
    "username": "my_username" // (use you preferred username)
}
```

In return you will receive a response with the following body:

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...."
}
```

Now copy that token (without the " "), and we'll use that to use the `/quotes/random` endpoint:

```
const out = document.getElementById("quote"); // Assuming an element with the id "quote"

const url = "https://api.noroff.dev/api/v1/quotes/random";
//console.log(url);

const options = {
    headers: {
        Authorization: "Bearer [USE YOUR TOKEN HERE]",
    }
};

getRandomQuote(url, options);

async function getRandomQuote(url, options) {
    console.log(url, options);
    const response = await fetch(url, options);
    console.log(response);
    const data = await response.json();
    //console.log(data);
    displayFact(data, out);
}

function displayFact({id, content, author}, outElement) {
    //console.log(id, content, author);
    outElement.innerHTML = `Random quote #${id}:<br><em>${content}</em><br>- ${author}`;
}
```

URL Parameters

URL

A **Uniform Resource Locator** (URL), colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.

A URL is a specific type of **Uniform Resource Identifier** (URI), although many people use the two terms interchangeably.

URLs occur most commonly to reference web pages (`HTTP`) but are also used for file transfer (`FTP`), email (`mailto`), database access (`JDBC`), and many other applications.

A typical URL could have the form http://www.example.com/index.html, which indicates a protocol (`http`), a hostname (`www.example.com`), and a file name (`index.html`).

Source Wikipedia: URL

URL Syntax

Every HTTP URL conforms to the syntax of a generic URI. The URI generic syntax consists of a hierarchical sequence of five components:[13]

```
URI = scheme ":" ["//" authority] path ["?" query] ["#" fragment]
```

where the authority component divides into three subcomponents:

```
authority = [userinfo "@"] host [":" port]
```

This is represented in a syntax diagram as:

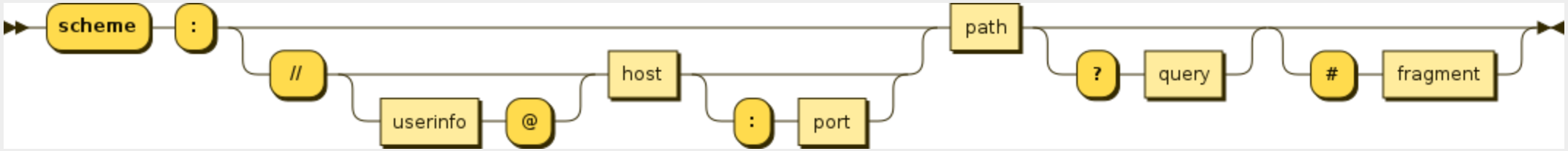
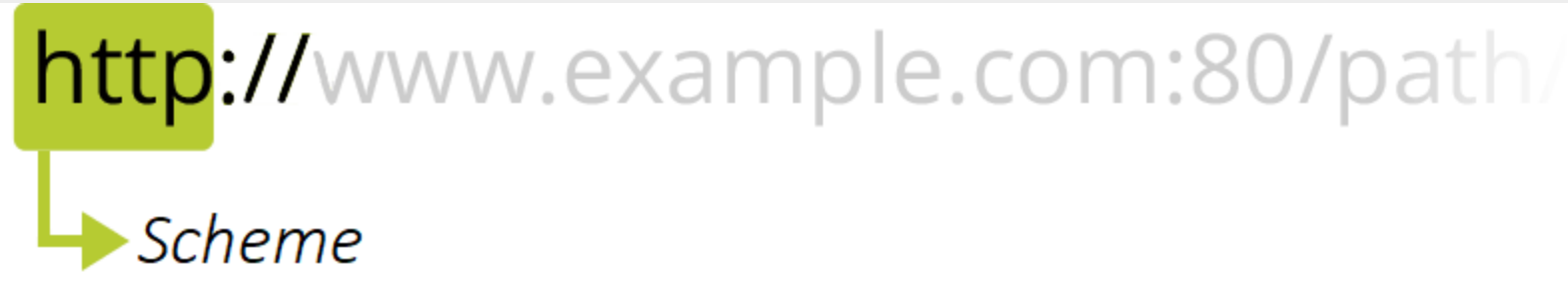


Diagram source

Scheme



The first part of the URL is the **scheme**, which indicates the protocol that the browser must use to request the resource (a protocol is a set method for exchanging or transferring data around a computer network).

Usually for websites the protocol is `HTTPS` or `HTTP` (its unsecured version).

Addressing web pages requires one of these two, but browsers also know how to handle other schemes such as `mailto` (to open a mail client), so don't be surprised if you see other protocols.

Source What is a URL?

Authority

Next follows the authority, which is separated from the scheme by the character pattern `://`. If present the authority includes both the domain (e.g. `www.example.com`) and the port (`80`), separated by a colon `:`

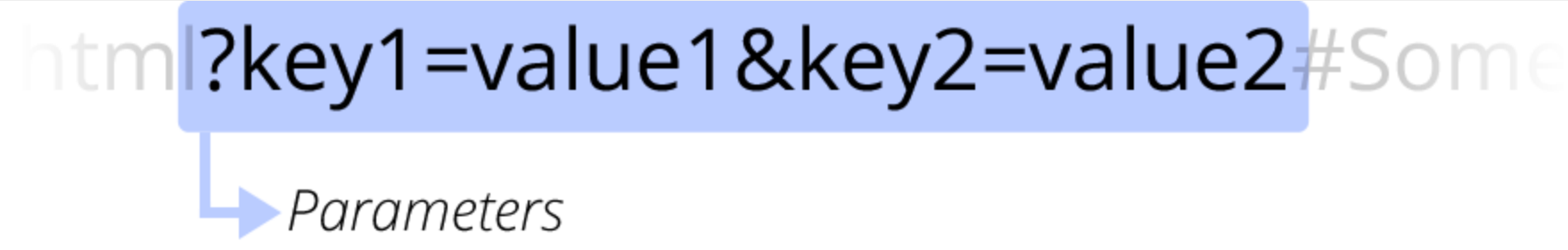
- The domain indicates which Web server is being requested. Usually this is a domain name, but an IP address may also be used (but this is rare as it is much less convenient).
- The port indicates the technical "gate" used to access the resources on the web server. It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources. Otherwise it is mandatory.

Path to resource

`/path/to/myfile.html` is the path to the resource on the Web server.

In the early days of the Web, a path like this represented a physical file location on the Web server. Nowadays, it is just as likely an abstraction handled by Web servers without any physical reality.

Parameters



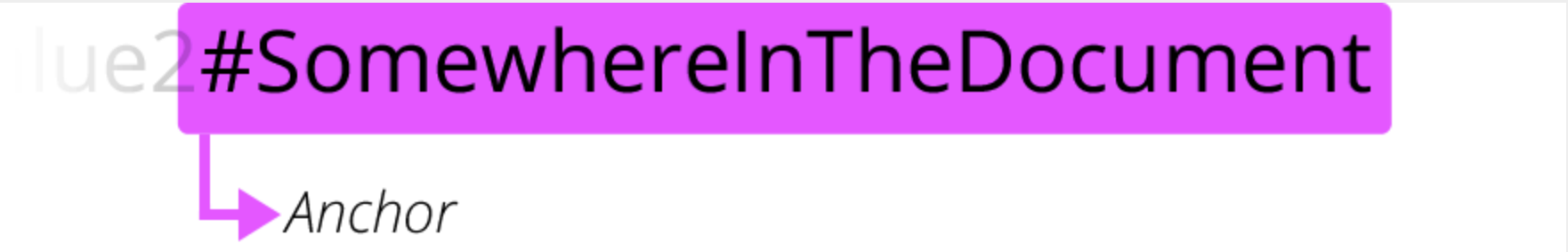
`?key1=value1&key2=value2` are extra parameters provided to the Web server.

Those parameters are a list of key/value pairs separated with the `&` symbol.

The Web server can use those parameters to do extra stuff before returning the resource.

Each Web server has its own rules regarding parameters, and the only reliable way to know if a specific Web server is handling parameters is by asking the Web server owner.

Anchor



`#SomewhereInTheDocument` is an anchor to another part of the resource itself.

An anchor represents a sort of "bookmark" inside the resource, giving the browser the directions to show the content located at that "bookmarked" spot.

On an HTML document, for example, the browser will scroll to the point where the anchor is defined; on a video or audio document, the browser will try to go to the time the anchor represents.

It is worth noting that the part after the #, also known as the fragment identifier, is never sent to the server with the request.

URLSearchParams

The `URLSearchParams` interface defines utility methods to work with the query string of a URL.

An object implementing `URLSearchParams` can directly be used in a `for...of` structure to iterate over key/value pairs in the same order as they appear in the query string, for example the following two lines are equivalent:

```
for (const [key, value] of mySearchParams) {}
for (const [key, value] of mySearchParams.entries()) {}
```

URLSearchParams

URLSearchParams()

The `URLSearchParams()` constructor creates and returns a new `URLSearchParams` object, using the URL.search property that represents the *query string* aka parameters of the URL:

```
const url = new URL('https://example.com?foo=1&bar=2');
const params1 = new URLSearchParams(url.search);
console.log(params1);
// > URLSearchParams { foo → "1", bar → "2" }
```

URLSearchParams.has()

The `has()` method of the `URLSearchParams` interface returns a boolean value that indicates whether a parameter with the specified name exists.

```
let url = new URL('https://example.com?foo=1&bar=2');
let params = new URLSearchParams(url.search);

console.log(params.has('bar')); //true
```

URLSearchParams.get()

The `get()` method of the `URLSearchParams` interface returns *the first value* associated to the given search parameter.

Requesting a parameter that isn't present in the query string will return `null`.

Example: If the URL of your page is `https://example.com/?name=Jonathan&age=18` you could parse out the '`name`' and '`age`' parameters using:

```
let params = new URLSearchParams(document.location.search);
let name = params.get("name");
let age = parseInt(params.get("age"), 10);
let address = params.get("address");

console.log (name, age, address); // "Jonathan", 18, null
```

URLSearchParams.getAll()

The `getAll()` method of the `URLSearchParams` interface returns *all the values* associated with a given search parameter as an array.

This is useful when you have several parameters with the same key:

```
const paramStr = 'foo=bar&foo=baz';
const searchParams = new URLSearchParams(paramStr);

console.log(searchParams.get('foo'));
// bar, as get() only returns the first value

console.log(searchParams.getAll('foo'));
// ["bar", "baz"]
```

URL: searchParams property

The `searchParams` read-only property of the `URL` interface returns a `URLSearchParams` object allowing access to the `GET` decoded query arguments contained in the URL.

We can use this to:

- retrieving parameters from the query string.
- passing variables to other pages in the query string.

If the URL of your page is `https://example.com/?name=Jonathan%20Smith&age=18` you could parse out the name and age parameters using:

```
let params = new URL(document.location).searchParams;
let name = params.get("name"); // is the string "Jonathan Smith".
let age = parseInt(params.get("age")); // is the number 18
```

Example

See `search-params.html`

HTML:

```
<ul>
    <li><a href="?">No parameters</a></li>
    <li><a href="?id=42">One parameter: id</a></li>
    <li><a href="?id=42&name=Marvin">One parameter: id and name</a></li>
    <li><a href="?q=my+search">One parameter: q</a></li>
    <li><a href="?test=blåbærsyltetøy">One parameter: test</a></li>
    <li><a href="?x=42&y=13&z=55">3 parameters: x, y and z</a></li>
</ul>
```

JavaScript:

```
// get the query string
const queryString = document.location.search;
// create an object that will allows us to access all the query string parameters
const mySearchParams = new URLSearchParams(queryString);

// List all query string parameters
for (const [key, value] of mySearchParams) {
    console.log(key, value);
}

// List value of id
console.log("id has the value: " + mySearchParams.get("id"));

// Note: All params are strings!
if (mySearchParams.get("x") && mySearchParams.get("y") && mySearchParams.get("z")) {
    x = mySearchParams.get("x");
    y = mySearchParams.get("y");
    z = mySearchParams.get("z");
    console.log ("x + y + z = " + (x + y + z));
    console.log ("x + y + z = " + (Number(x) + Number(y) + Number(z)));
}
```

Demo: Amiibo revisited, now with filters

Todos

Postman

Intro to Postman | Part 2: Authorize a Request (13:17)

Mollify

Read CORS**

Read JWT

Read URL Parameters, and do the Lesson Task

(** may be wrongly named as `HTTP POST Request Method` in the menu)