W05 Readings

**Fetch: Cross-Origin Requests**

Sending a fetch request to another website will most likely fail.

The core concept is origin - domain/port/protocol

Here is an example of (domain/protocol/port): Origin: https://javascript.info

The policy of “CORS”: Cross-Origin Resource Sharing, requires special headers from the remote side when you need Cross-origin requests to be sent to another domain, protocol, or port.

CORS protects you from hackers.

You can submit a <form> to communicate to another server.

You can also use a <script> tag to generate the data the remote server wants you to use.

The two types of cross-origin requests are:

1. Safe requests.
2. All the others.

Requests are safe when using two conditions:

1. Safe method: GET, POST or HEAD
2. Safe 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.

All other requests are unsafe.

You use fetch() in JavaScript to the browser > Origin is sent with a cross-origin request to the server > The server checks for permitting Access-Control-Allow-Origin in the response. If it’s allowed, JS is allowed access to the response.

JS may only access “safe” response headers for cross-origin request:

* Cache-Control
* Content-Language
* Content-Length
* Content-Type
* Expires
* Last-Modified
* Pragma

**Single-Page Applications**

single-page application (SPA), re-renders contents on a webpage in response to actions when navigating without fetching new HTML.

You don’t have to parse the interactive parts of a webpage when you use window.location for your SPA.

SPA’s rely on routers.

Routers describe the location that they should match.

Example:

const routes = [

{ path: '/' },

{ path: '/about' }, (This is static)

{ path: '/album/:id' } (This is dynamic)

];

After the router matches the route, it will re-render the page.

The History API updates the active document to the new location.

The History API has three functions:

 pushState() – Takes three arguments:

* state – Pass null is you don’t want to use state. State represents navigation.
* Title – No browsers use this yet.
* Path – The URL, an absolute path, or a relative path

history.pushState() adds an entry to the session history.

replaceState() – has the same three arguments as  pushState()

history.replaceState() replaces the current entry in the session history

go() – is similar to performing the browser’s forward and back buttons. It takes a single number as an argument:

go(-1); // go back one entry

go(1); // go forward one entry

go(-10); // go way back

go(0); // reload

go(); // reload