Week 5 WDD330

viernes, 19 de mayo de 2023 15:16

Rute Cordova!

Fetch

FETCH() MODULE IS USE FOR SEND NETWOR REQUESTS TO THE SERVER

- ```
  let promise = fetch(url, [options])
  ```

  URL TO ACCESS

  OPTIONAL PARAMETERS → DEFAULT IS A SIMPLE GET REQUEST

- RESPONSE: 1)- A PROMISED RETURNED BY FETCH
  - RESOLVES WITH AN OBJECT OF THE BUILT-IN RESPONSE CLASS

  (BOOLEAN) RESPONSE.OK → IF HTTP-STATUS IS 200-299

  RESPONSE.STATUS → HTTP-STATUS CODE

```
let response = await fetch(url);

if (response.ok) { // if HTTP-status is 200-299
  // get the response body (the method explained belo
  let json = await response.json();
} else {
  alert("HTTP-Error: " + response.status);
}
```

2)- TO GET THE RESPONSE BODY WE USE AN ADDITIONAL METHOD CALL

- `response.text()` – read the response and return as text,
- `response.json()` – parse the response as JSON,
- `response.formData()` – return the response as `FormData` object (explained in the next chapter),
- `response.blob()` – return the response as Blob (binary data with type),
- `response.arrayBuffer()` – return the response as ArrayBuffer (low-level representation of binary data),
- additionally, `response.body` is a ReadableStream object, it allows you to read the body chunk-by-chunk, we'll see an example later.

EXAMPLE WITHOUT AWAIT

```
fetch('https://api.github.com/repos/javascript-tutorial/en.javascript.info/commits')
  .then(response => response.json())
  .then(commits => alert(commits[0].author.login));
```
```

⚠ WE CAN ONLY USE ONE BODY READING METHOD

RESPONSE HEADERS

- USING RESPONSE.HEADERS OBJECT.

```
// get one header
alert(response.headers.get('Content-Type'));

// iterate over all headers
for (let [key, value] of response.headers) {
  alert(`${key} = ${value}`);
}
```

REQUEST HEADERS

- USING THE HEADERS OPTIONS.

```
let response = fetch(protectedUrl, {
  headers: {
    Authentication: 'secret'
  }
});
```

- THE LIST OF FORBIDDEN HTTP HEADERS: https://fetch.spec.whatwg.org/#forbidden-header-name

POST REQUESTS

- WE NEED TO USE FETCH OPTIONS

```
let response = await fetch('/article/fetch/post/user', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  body: JSON.stringify(user)
});
```

↳ THE REQUEST BODY COULD BE:

- A STRING (JSON-ENCODED) (MOST USED)
- FORMDATA OBJECT
- BLOB / BUFFER SOURCE
- URLSEARCHPARAMS

SENDING AN IMAGE:

- BINARY DATA WITH BLOB OR BUFFERSOURCE OBJECTS.

```
async function submit() {
  let blob = await new Promise(resolve => canvasElem.toBlob(resolve, 'image/png'));
  let response = await fetch('/article/fetch/post/image', {
    method: 'POST',
    body: blob
  });
```

- WE DON'T SET CONTENT-TYPE HEADER MANUALLY BECAUSE A BLOB OBJECT HAS A BUIT-IN TYPE

CORS: CROSSING-ORIGIN REQUESTS

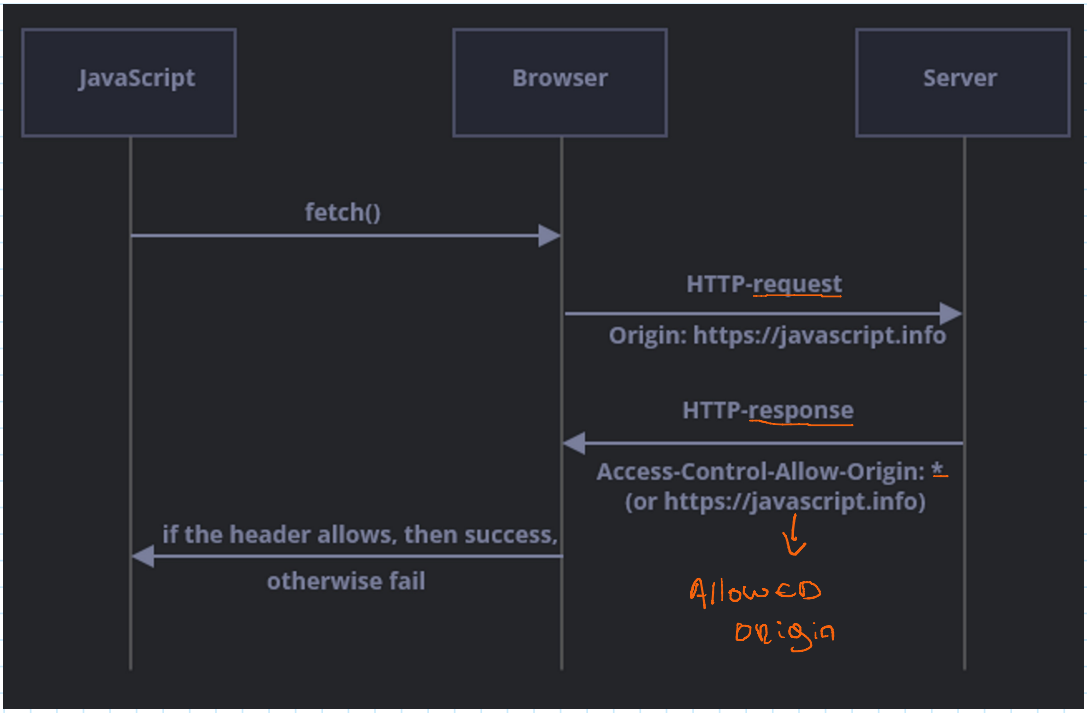
↳ CROSS-ORIGIN RESOURCE SHARING: A POLICY BETWEEN REQUESTS TO ANOTHER WEB-SITE

- THERE ARE TWO TYPES OF CROSS-ORIGIN REQUESTS:
  1. SAFE REQUESTS (SAFE METHOD, SAFE HEADERS)
  2. ALL THE OTHERS

1. **SAFE REQUESTS**: IT SATISFIES 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`.

- IT CAN BE MADE WITH A <FORM> OR A <SCRIPT> WITHOUT ANY SPECIAL METHOD
- THE BROWSER ALWAYS ADDS THE ORIGIN HEADER TO THE REQUEST.
- THE SERVER INSPECT THE ORIGIN AND IF ITS AGREE, ADDS A SPECIAL HEADER TO THE RESPONSE ACCESS-CONTROL-ALLOW-ORIGIN



REQUEST:

```
GET /request
Host: anywhere.com
Origin: https://javascript.info
...
```

RESPONSE:

```
200 OK
Content-Type:text/html; charset=UTF-8
Access-Control-Allow-Origin: https://javascript.info
```

- SAFE RESPONSE HEADERS:

- ACCESSING ANY OTHER RESPONSE HEADER CAUSES AN ERROR

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

- TO USE OTHER HEADERS WE MUST SEND `Access-Control-Expose-Headers` HEADER WITH THE LIST OF UNSAFE HEADERS.

```
Access-Control-Expose-Headers: Content-Encoding,API-Key
```

② UNSAFE REQUESTS:

- THE BROWSER WILL MAKE A "PREFLIGHT" (PREVIOUS) REQUEST FOR UNSAFE REQUESTS

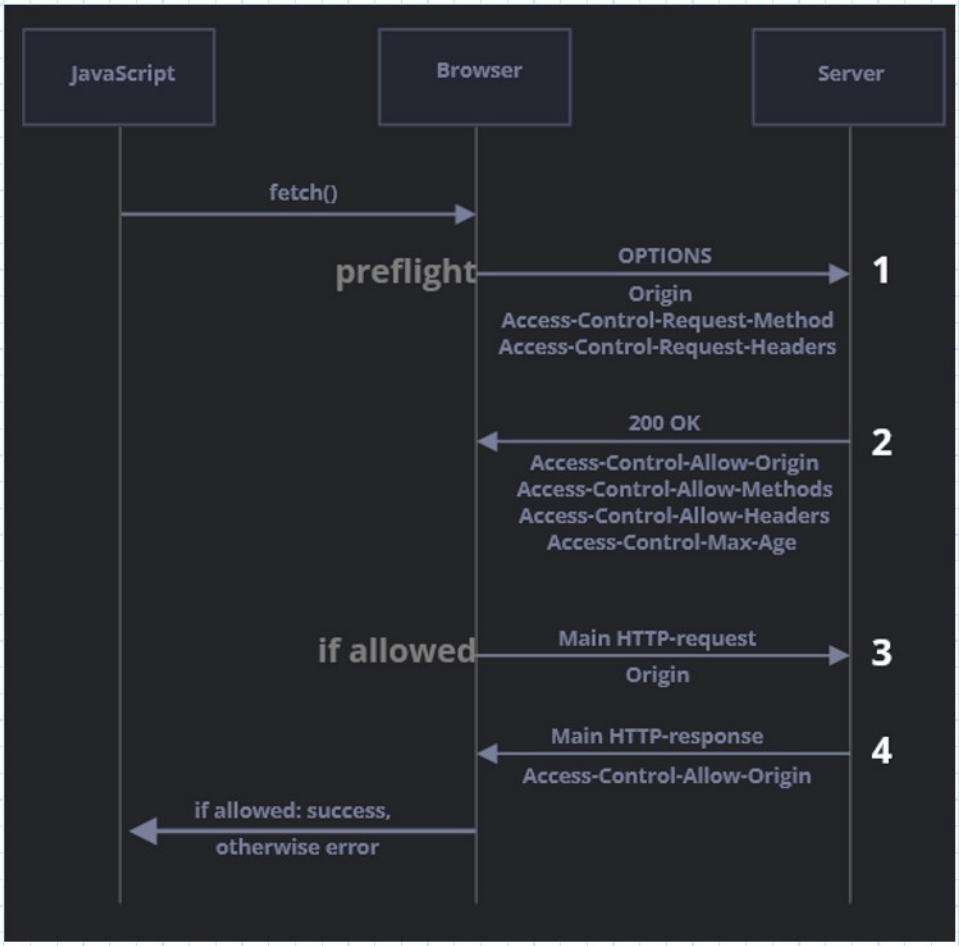
- THE BROWSER WILL MAKE A "PREFLIGHT" REQUEST FOR UNSAFE REQUESTS

PREFLIGHT REQUESTS USES THE METHOD OPTIONS, NO BODY AND 3 HEADERS

1 - OPTIONS
2 - ACCESS-CONTROL-REQUEST-METHOD
3 - ACCESS-CONTROL-REQUEST-HEADERS
4 - ORIGIN

- IF SERVER AGREES, THE RESPONSE GIVES:

1 - ACCESS-CONTROL-ALLOW-ORIGIN
2 - ACCESS-CONTROL-ALLOW-METHODS
3 - ACCESS-CONTROL-ALLOW-HEADERS
4 - ACCESS-CONTROL-MAX-AGE → NUM OF SEC. TO CACHE THE PERMISSIONS



PREFLIGHT REQUEST:

```
1 OPTIONS /service.json
2 Host: site.com
3 Origin: https://javascript.info
4 Access-Control-Request-Method: PATCH
5 Access-Control-Request-Headers: Content-Type,API-Key
```

PREFLIGHT RESPONSE: (IF AGREES)

```
200 OK
Access-Control-Allow-Origin: https://javascript.info
Access-Control-Allow-Methods: PUT,PATCH,DELETE
Access-Control-Allow-Headers: API-Key,Content-Type,If-Modified-Since,Cache-Control
Access-Control-Max-Age: 86400
```

IF PREFLIGHT IS SUCCESSFUL, THE MAIN REQUEST IS DONE

REQUEST :

```
PATCH /service.json
Host: site.com
Content-Type: application/json
API-Key: secret
Origin: https://javascript.info
```

RESPONSE :

```
Access-Control-Allow-Origin: https://javascript.info
```

CREDENTIALS: (COOKIES OR HTTP AUTHENTICATION)

- IF ALLOWED, IT GRANTS JS THE FULL POWER TO ACT ON BEHALF OF THE USER AND ACCESS SENSITIVE INFO USING THEIR CREDENTIALS.
- TO SEND CREDENTIALS WE WRITE:

```
fetch('http://another.com', {
  credentials: "include"
});
```

RESPONSE :

```
200 OK
Access-Control-Allow-Origin: https://javascript.info
Access-Control-Allow-Credentials: true
```

REST PARAMETERS AND SPREAD SYNTAX

- BOTH OF THEM ARE WRITE WITH 3 DOTS(...)
- WHEN ... IS AT THE END OF FUNCTION PARAMETERS, ITS REST PARAMETERS
  - IT GATHERS THE REST OF THE LIST OF ARGUMENTS INTO AN ARRAY

```
function sumAll(...args) { // args is the name for the array
  let sum = 0;

  for (let arg of args) sum += arg;

  return sum;
}

```

```
function sumAll(...args) { // args is the name for the array
  let sum = 0;

  for (let arg of args) sum += arg;

  return sum;
}

alert( sumAll(1) ); // 1
alert( sumAll(1, 2) ); // 3
alert( sumAll(1, 2, 3) ); // 6
```

- ARE USED TO CREATE FUNCTIONS THAT ACCEPT ANY NUMBER OF ARGUMENTS

- WHEN ... IS IN A FUNCTION CALL OR ALIKE, IT'S CALLED A SPREAD SYNTAX
  - IT EXPANDS AN ARRAY INTO A LIST

```
let arr = [3, 5, 1];

alert( Math.max(...arr) ); // 5 (spread turns array into a list of arguments)
```

```
let str = "Hello";

alert( [...str] ); // H,e,l,l,o
```

- IT'S USED TO PASS AN ARRAY TO FUNCTIONS THAT REQUIRE A LIST OF MANY ARGUMENTS.

COPY AN ARRAY/OBJECT .

```
let arr = [1, 2, 3];

let arrCopy = [...arr]; // spread the array into a list of parameters
                        // then put the result into a new array

// do the arrays have the same contents?
alert(JSON.stringify(arr) === JSON.stringify(arrCopy)); // true

// are the arrays equal?
alert(arr === arrCopy); // false (not same reference)

// modifying our initial array does not modify the copy:
arr.push(4);
alert(arr); // 1, 2, 3, 4
alert(arrCopy); // 1, 2, 3
```

```js
let obj = { a: 1, b: 2, c: 3 };

let objCopy = { ...obj }; // spread the object into a list of parameters
                          // then return the result in a new object

// do the objects have the same contents?
alert(JSON.stringify(obj) === JSON.stringify(objCopy)); // true

// are the objects equal?
alert(obj === objCopy); // false (not same reference)

// modifying our initial object does not modify the copy:
obj.d = 4;
alert(JSON.stringify(obj)); // {"a":1,"b":2,"c":3,"d":4}
alert(JSON.stringify(objCopy)); // {"a":1,"b":2,"c":3}
```

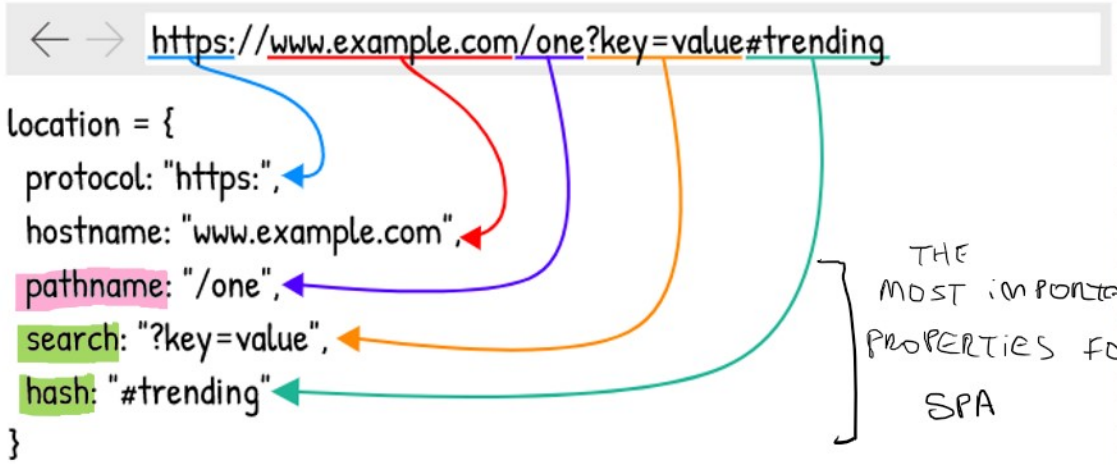
SINGLE-PAGE APPLICATIONS (SPA)

- IT'S A WEBSITE THAT RENDERS ITS CONTENT IN RESPONSE TO NAV ACTIONS WITHOUT MAKING A REQUEST TO THE SERVER TO FETCH NEW HTML.
- USE A SINGLE HTML FILE (PAGE), BUT NOT MULTIPLE-PAGES AND ITS CONTENT CHANGES THROUGH NAVIGATION
- IT'S EXECUTED BY THE BROWSER.

INTERNAL STATE SPA :
- CAN ONLY LOAD THE APP'S ENTRY
- NAV DOES NOT UPDATE LOCATION (URL)
- YOU'LL HAVE TO EXPLAIN HOW TO GET THE DESIRE CONTENT

LOCATION-BASED SPA
- NAV UPDATES THE LOCATION
- CAN IMMEDIATELY RENDER THE DESIRED CONTENT
- IF YOU SHARE THE LINK, IT WILL LOAD SAME CONTENT YOU ARE WATCHING

SPA USE WINDOW.LOCATION TO MAP FROM TH URL



PATHNAME DETERMINES WHAT CONTENT TO RENDER

SPA GENERALLY RELY ON A ROUTER

```
const routes = [
  { path: '/' },
  { path: '/about' },
  { path: '/album/:id' }
];
```

DIFFERENT ROUTES

STATIC

DYNAMIC

- WHEN MATCHING A ROUTE, THE ROUTER WILL TRIGGER A RE-RENDER OF THE APP.

IN-APP NAV :
- IS PERFORMED USING THE HISTORY API
- THE DEFAULT BEHAVIOR FOR <a> IS OVERRIDEN TO USE PUSHSTATE() AND REPLACESTATE() TO NAVIGATE
- A POPSTATE EVENT LISTENER IS USE TO DETECT NAVIGATION (FOWARD & BACK BUTTONS)
- THE EVENT INFORMS THE ROUTER TO MATCH THE NAV PATH

- PUSHSTATE() & REPLACESTATE():

(STATE, TITLE, PATH)

```
history.pushState(null, '', '/next-location');
history.replaceState(null, '', '/replace-location');
// attaching state to an entry
history.pushState({ msg: 'Hi!' }, '', '/greeting');
// while on medium.com
history.pushState(null, '', 'https://www.google.com');
```

```
history.pushState(null, '', '/next-location');
history.replaceState(null, '', '/replace-location');
// attaching state to an entry
history.pushState({ msg: 'Hi!' }, '', '/greeting');
// while on medium.com
history.pushState(null, '', 'https://www.google.com');
// throws a DOMException
```

PUSHSTATE() ADDS AN ENTRY TO THE SESSION HISTORY AFTER THE CURRENT ENTRY

REPLACESTATE() REPLACES THE CURRENT ENTRY IN THE SESSION HISTORY

```
// React example
const Link = ({ children, href }) => (
  <a
    href={href}
    onClick={event => {
      // override native behavior
      event.preventDefault();
      // navigate using the History API
      // (ignoring replaceState() for brevity)
      history.pushState(null, '', href);
      // finally, let the router know navigation happened!
    }
  >
    {children}
  </a>
);
<Link href="/somewhere">Somewhere</Link>
// renders
<a href="/somewhere">Somewhere</a>
// but clicking on it will trigger a history.pushState() call
```

```
window.addEventListener('popstate', event => {
  // let the router know navigation happened!
}, false);
```

Q: HOW DO YOU KNOW A PAGE WAS MADE WITH SPA?
I REFFER WHEN BROWSING IN THE WEB

DISVANTAGES:
- YOU END UP RECREATING WITH JS A LOT OF THE FEATURES THE BROWSER ALREADY HAS.
- MORE CODE TO MAINTAIN AND COMPLEXITY
- MORE THINGS TO BREAK

Q: IT FEELS LIKE RE-INVENTING THE WHEEL, BUT WHEN IS A GOOD EXAMPLE TO IMPLEMENT THE SPA?
↓
OR MOMENT