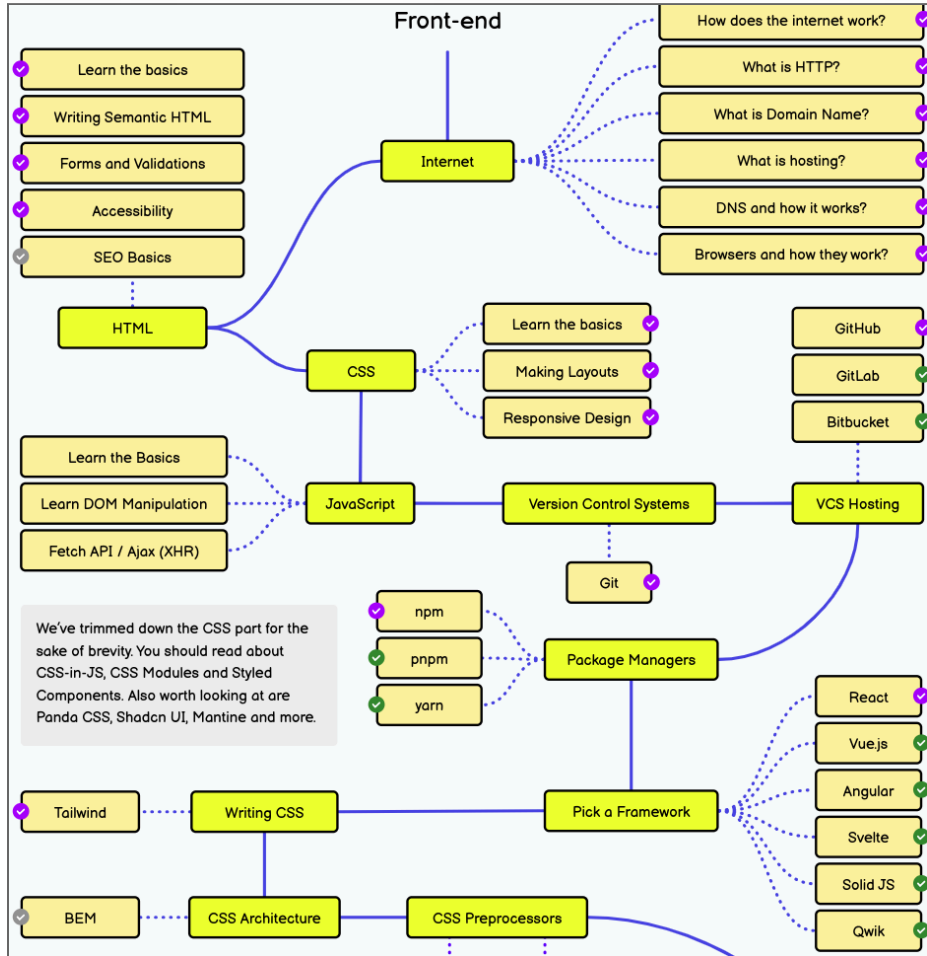


PROYECTO DE SOFTWARE

SEGUIMOS CON EL PROCESAMIENTO EN EL CLIENTE



• Fuente: <https://roadmap.sh/frontend>

TEMARIO

- Peticiones asincrónicas:
 - AJAX
 - Fetch API
 - Async/Await
- CORS

PETICIONES ASINCRÓNICAS

AJAX: ASYNCHRONOUS JAVASCRIPT + XML



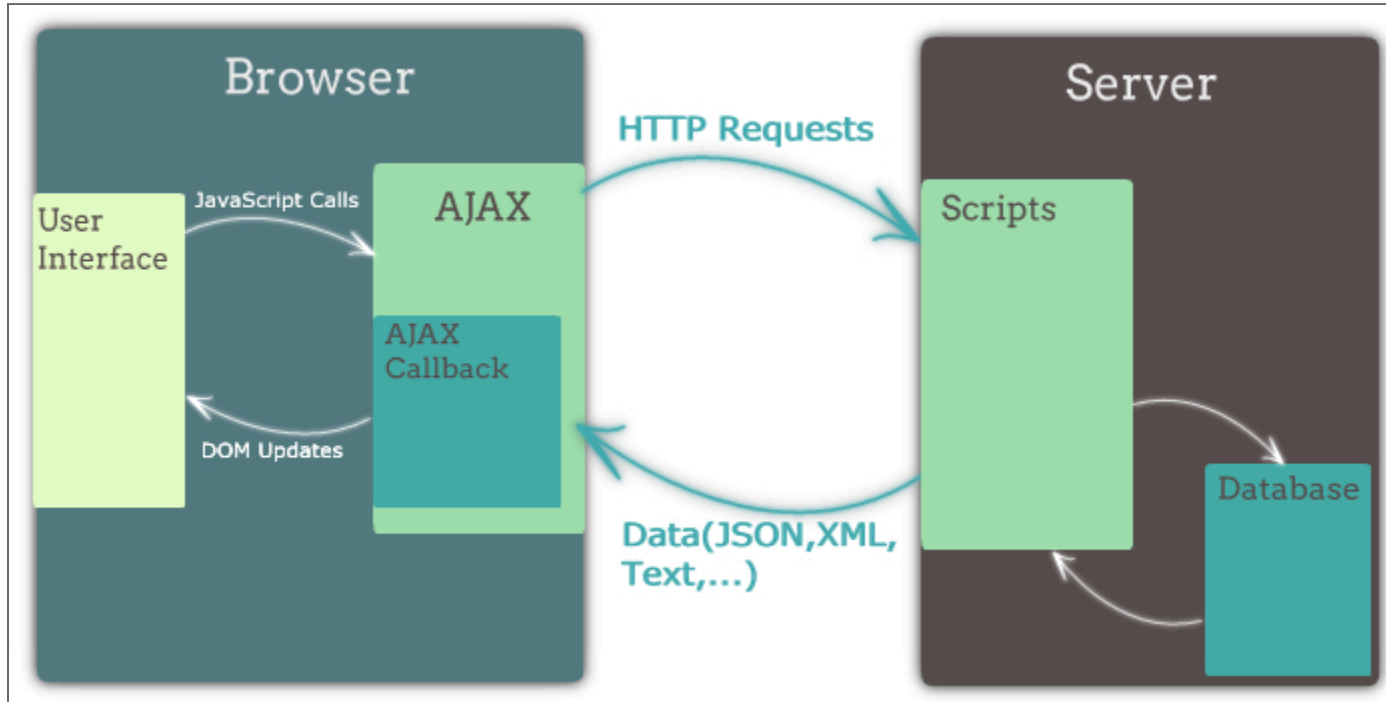
AJAX

- NO es una tecnología, sino una combinación de varias tecnologías.
- AJAX incluye:
 - Presentación basada en estándares usando **HTML** y **CSS**;
 - Exhibición e interacción dinámicas usando **DOM**;
 - Intercambio y manipulación de datos usando **XML** y **XSLT**;
 - Nosotros usaremos JSON.
 - Recuperación de datos asincrónica usando **XMLHttpRequest**;
 - **JavaScript** como lenguaje de programación.

AJAX

- Comenzó a ser popular a partir del año 2005, con Google Suggest.
- El objetivo es crear interfaces de usuario más amigables, similares a las de las PC de escritorio, sin afectar los tiempos y el esquema de navegación.
- ¡¡**IMPORTANTE!!** El feedback al usuario.

FUNCIONAMIENTO AJAX



EL OBJETO XMLHttpRequest

- Es un objeto que permite realizar requerimientos HTTP al servidor web desde cualquier lenguaje de script client-side SIN recargar la página.
- La especificación en Web Hypertext Application Technology Working Group (WHATWG).

EL OBJETO XMLHttpRequest (CONT.)

- Algunas propiedades...
- **onreadystatechange**: manejador de evento para un cambio de estado.
- **readyState**: el estado del objeto:
 - 0 = UNSENT
 - 1 = OPENED
 - 2 = HEADERS_RECEIVED
 - 3 = LOADING
 - 4 = DONE
- A partir de Level 2 se definieron más eventos/manejadores

EL OBJETO XMLHttpRequest (CONT.)

- Algunas propiedades (cont.)...
 - **responseText**: retorna la respuesta como texto.
 - **responseXML**: retorna la respuesta como XML que puede ser manipulado usando DOM.
- Algunos métodos...
 - **open("method", "URL", async, "uname", "pswd")**: especifica el método, URL y otros atributos opcionales del requerimiento:
 - El método puede ser "GET", "POST", o "PUT"
 - La URL puede ser una URL completa o relativa
 - El parámetro **async** especifica si el requerimiento debe ser manejado en forma asincrónica o no (true o false)

EJEMPLOS UTILIZANDO XMLHTTPREQUEST (ASYNC)

- AJAX de la forma tradicional en forma asincrónica:

```
function buscarAsync() {
    xhr = new XMLHttpRequest();
    var param = document.getElementById('interprete').value;
    var url = "http://localhost:5000/musicos?name=" +
escape(param);
    xhr.open("GET", url, true);
    xhr.onreadystatechange = cargoInfo;
    xhr.send();
}
function cargoInfo() {
    document.getElementById('readyState').textContent =
xhr.readyState;
    document.getElementById('status').textContent = xhr.status;
    if (xhr.readyState == 4)
    {
        if (xhr.status == 200) {
            rta_json = JSON.parse(xhr.responseText);
            if (rta_json.musician){
                document.getElementById('info').textContent =
rta_json.musician.description;
            }
            else {
                document.getElementById('info').textContent = "No
encontrado";
            }
        }
        else alert("Algo anda mal");
    }
}
```

```
}  
}
```

- Ver ejemplo ajax asincrónico

EJEMPLOS UTILIZANDO XMLHttpRequest (SYNC)

- AJAX de la forma tradicional en forma sincrónica (DEPRECATED):

```
function buscarSync() {
    xhr = new XMLHttpRequest();
    var param = document.getElementById('interprete').value;
    var url = "http://localhost:5000/musicos?name=" +
escape(param);
    xhr.open("GET", url , false);
    xhr.send();

    if (xhr.status == 200) {
        rta_json = JSON.parse(xhr.responseText);
        if (rta_json.musician){
            document.getElementById('info').textContent =
rta_json.musician.description;
        }
        else {
            document.getElementById('info').textContent = "No
encontrado";
        }
    }
}
```

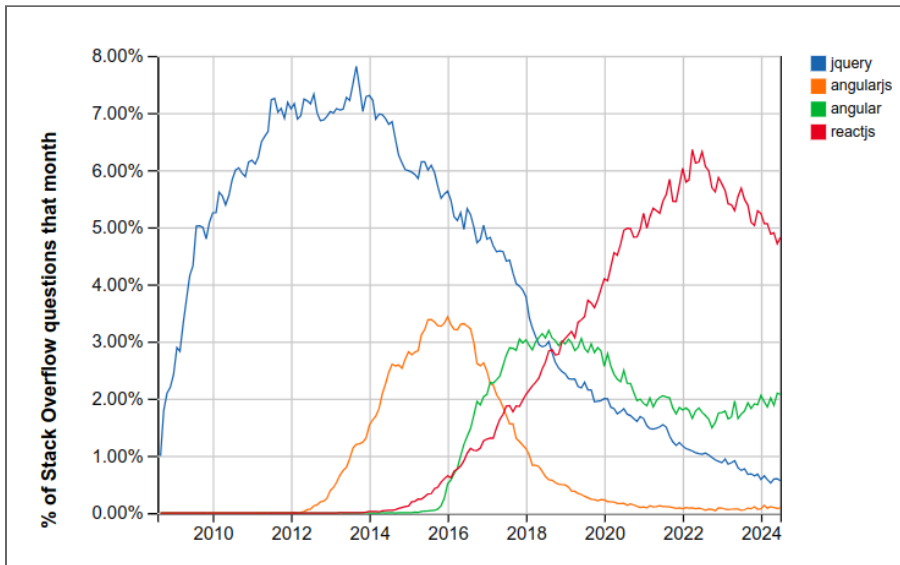
- Ver [ejemplo ajax sincrónico](#)

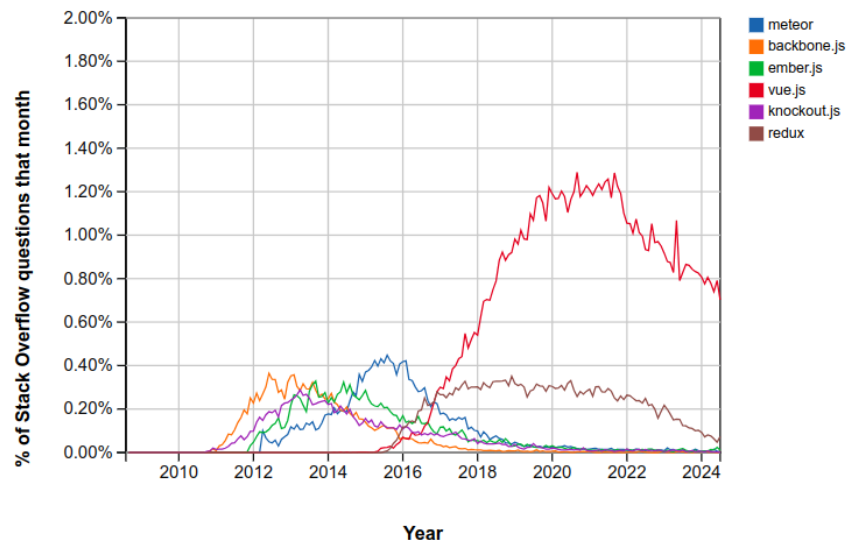
AJAX CON JQUERY



ANTES HABLEMOS DE LIBRERÍAS/Frameworks JAVASCRIPT

- Contienen soluciones ya implementadas, sólo debemos usarlas.
- El objetivo es **simplificar el desarrollo**. Pero... **¡Hay muchas!**
- Todos los años aparecen nuevas. Ver artículo...
- Las más consultadas según Stackoverflow agrupadas en frameworks JS y smaller frameworks JS.





LIBRERÍAS/Frameworks JAVASCRIPT

- Tendencias de JS en Github.
- Las librerías más utilizadas y JavaScripting.
- **No todas con el mismo objetivo.**
- Para desarrollo en los últimos años...



¿JQUERY EN LA ACTUALIDAD?

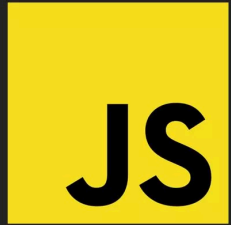
Para desarrollos sencillos

- Aún actualmente, muy usada:
https://w3techs.com/technologies/overview/javascript_library/all
- Atajos a las funciones de DOM:
 - `document.getElementById("p1")` vs. `$("#p1")`
 - `document.getElementsByTagName("p")` vs. `$("p")`
- JQuery usa los selectores CSS para acceder a los elementos:
 - `$("p.intro")`: todos los elementos `<p>` con `class="intro"`.
 - `$(".intro")`: todos los elementos con `class="intro"`
 - `$("p#demo")`: todos los elementos `<p>` `id="demo"`.
 - `$(this)`: el elemento actual
 - `$("ul li:odd")`: los `` impares dentro de ``

JQUERY: AJAX

- Veamos un ejemplo de ajax con JQuery

```
$.ajax({  
  url: '/ruta/hasta/pagina',  
  type: 'POST',  
  async: true,  
  data: 'parametro1=valor1&parametro2=valor2',  
  success: procesaRespuesta,  
  error: muestraError  
});
```



**FETCH
API**

FETCH API

- Introducidas en ECMAScript 2015(ES6).
- La API Fetch proporciona una interfaz para recuperar recursos.
- Fetch ofrece una definición genérica de los objetos **Request** y **Response**.
- El método **fetch()** toma un argumento obligatorio, la ruta de acceso al recurso que desea recuperar.
- Devuelve una **Promise** que resuelve en **Response** a esa petición, sea o no correcta.
- Es el reemplazo natural del objeto **XMLHttpRequest**.

PROMESAS JS

Una **Promise** es un objeto que representa la eventual finalización o falla de una operación asíncrona. (Ref. **Promise**)



Promise



Pending



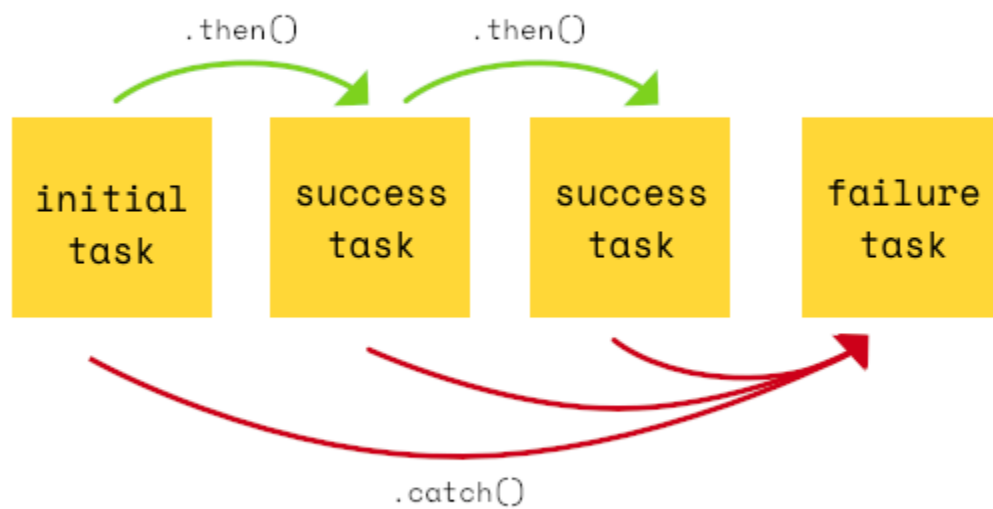
Resolved



Rejected

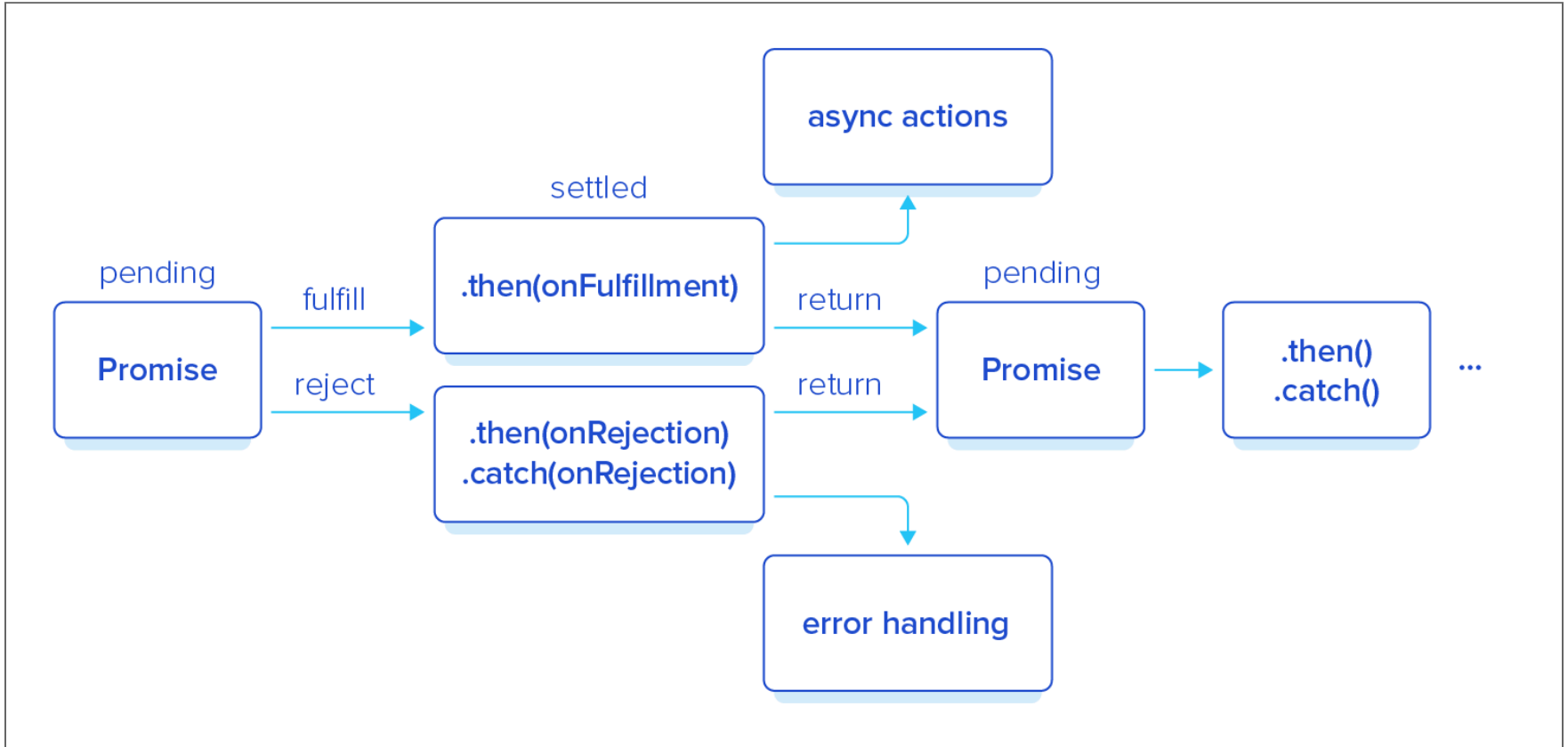
FETCH API

- Como **fetch()** devuelve una **promise**, esta puede encadenarse utilizando varios **.then()** y un **.catch()** si alguna falla.
- Esto nos permite establecer **lógica entre varios requerimientos**.



PROMESAS JS

- Flujo de una promesa en Javascript y su concatenación:



FETCH API

- Veamos un ejemplo: http://localhost:5000/ejemplo_ajax_fetch

```
function checkStatus(response) {
  if (response.status >= 200 && response.status < 300) {
    return Promise.resolve(response)
  } else {
    return Promise.reject(new Error(response.statusText))
  }
}

function parseJson(response) {
  return response.json()
}

function buscarFetch(){
  fetch('http://localhost:5000/all_musicos')
    .then(checkStatus)
    .then(parseJson)
    .then(function(data) {
      console.log('Request succeeded with JSON response',
data);
      document.getElementById('info').textContent =
JSON.stringify(data.musician);
    }).catch(function(error) {
      console.log('Request failed', error);
      document.getElementById('error').textContent = error;
    });
}
```

- Referencia Api Fetch

JS Async/await

FUNCIONES ASYNC

- Introducidas en ECMAScript 2017(ES8), las funciones **async** facilitan trabajar con promesas.
- Define una función **asincrónica** que utiliza una **Promise** para retornar su resultado.

OPERADOR AWAIT

- El operador **await** es utilizado para esperar por un **Promise**.
- Sólo puede ser utilizada dentro de una función **async**.
- Causa que la función **async** quede **pausada** hasta que la promesa se resuelva.

EJEMPLO ASYNC/AWAIT CON FETCH

- Veamos un ejemplo: http://localhost:5000/ejemplo_ajax_async_await

```
async function getPostsAsync()
{
  let response = await
fetch(`https://jsonplaceholder.typicode.com/posts`);
  let data = await response.json()
  document.getElementById('info').textContent =
JSON.stringify(data);
}
```

- Referencia async y await.

¿ALGUNO SABE QUÉ ES **CORS**?

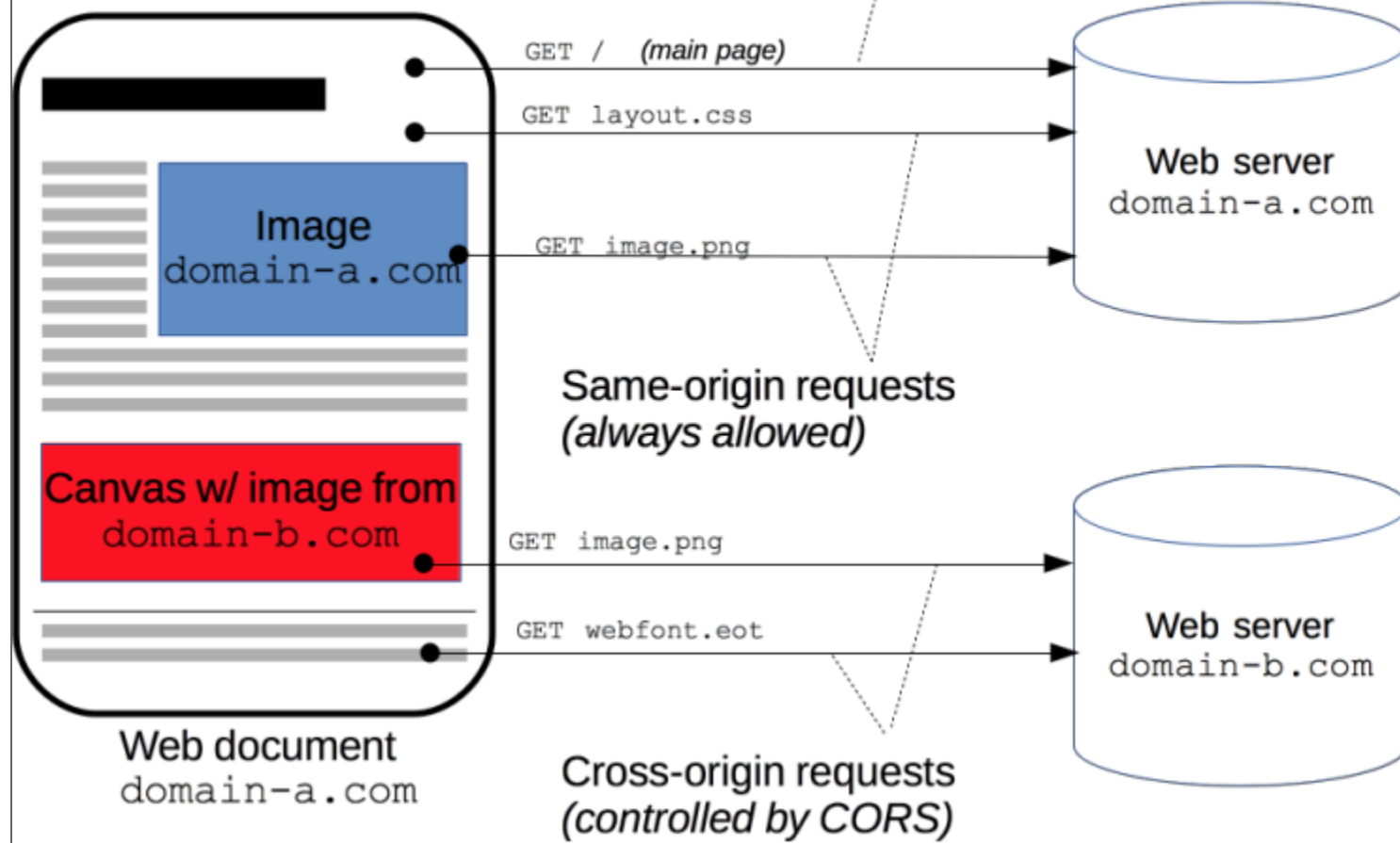
The logo for CORS (Cross-Origin Resource Sharing) is displayed within a thin black rectangular border. It features the word "CORS" in a bold, sans-serif font. The letters are color-coded: the 'C' is orange, the first 'O' is dark grey, the 'R' is blue, and the 'S' is dark grey. The letters are slightly overlapping, with the 'R' positioned between the 'O' and the 'S'.

CORS

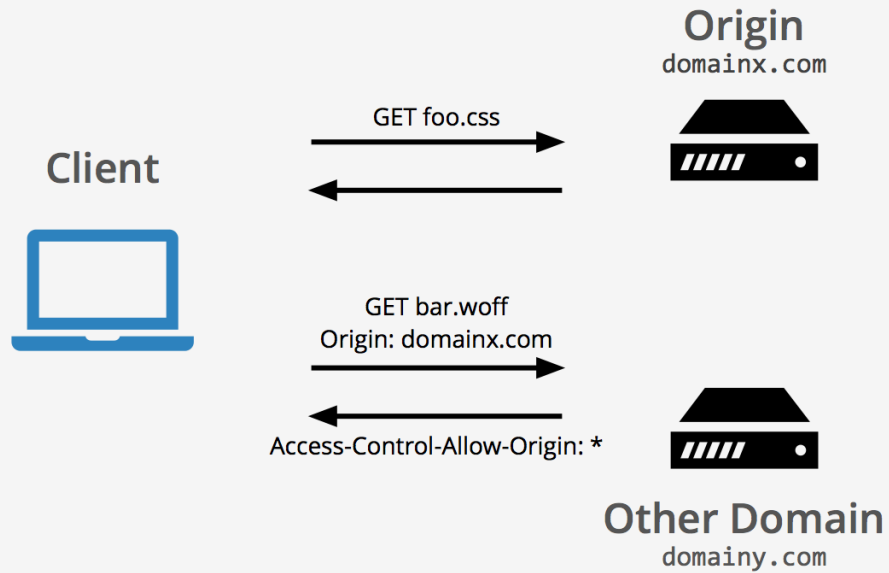
- **Cross-Origin Resource Sharing** (CORS)
- **CORS** es un mecanismo para permitir realizar peticiones de dominios cruzados utilizando Javascript.
- Por defecto **los navegadores actuales bloquean estas peticiones** si no se encuentran bien configurados tanto los clientes como los servidores.

CORS

Main request: defines origin.

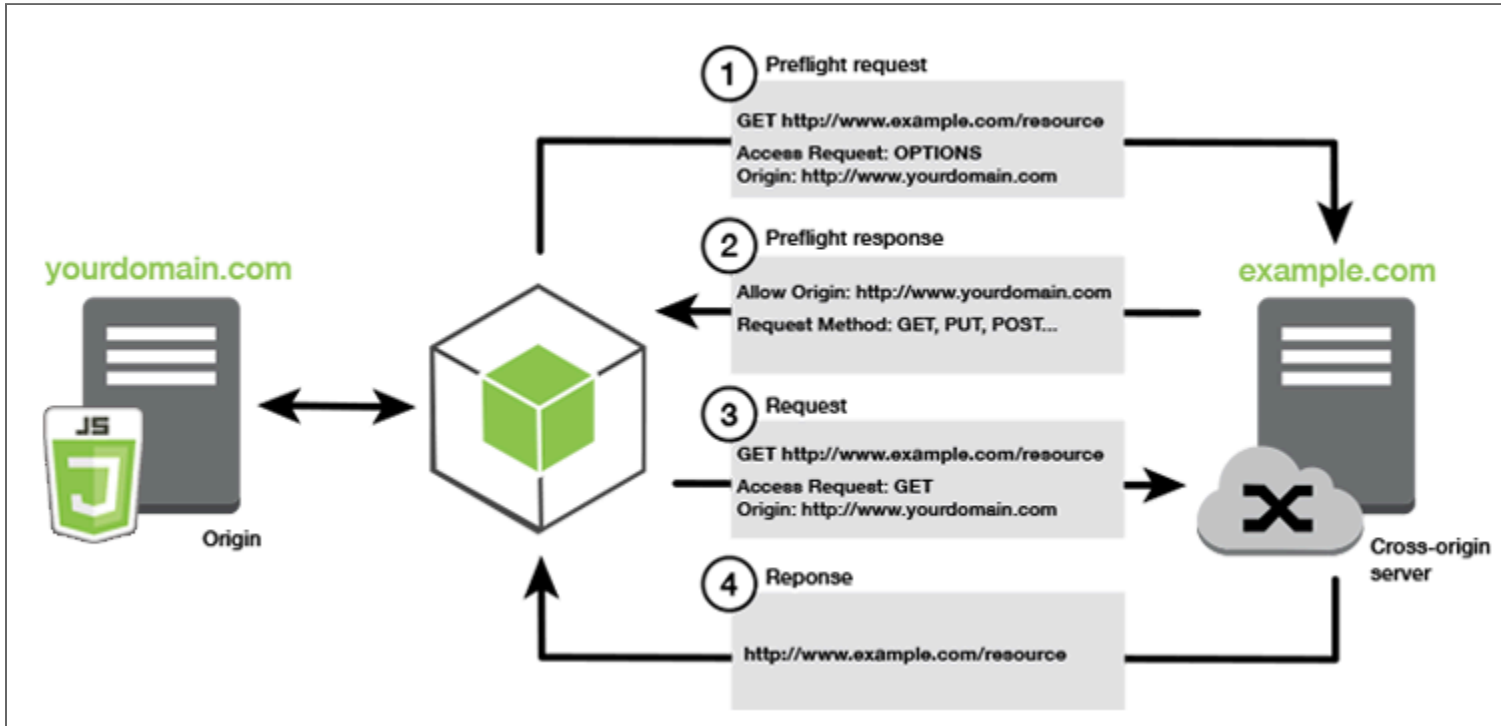


EL CASO MÁS SIMPLE



CORS

DIAGRAMA COMPLETO



REFERENCIAS CORS

- <https://enable-cors.org/>
- [https://developer.mozilla.org/es/docs/Web/HTTP/Access control CORS](https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS)

ALGUNAS GUÍAS DE ESTILO JAVASCRIPT

- Crockford: <https://www.crockford.com/code.html>
- Airbnb: <https://github.com/airbnb/javascript>
- Google: <https://google.github.io/styleguide/jsguide.html>
- Standard: <https://standardjs.com/>

¿DUDAS?

SEGUIMOS LA PRÓXIMA ...

Speaker notes