PROYECTO DE SOFTWARE

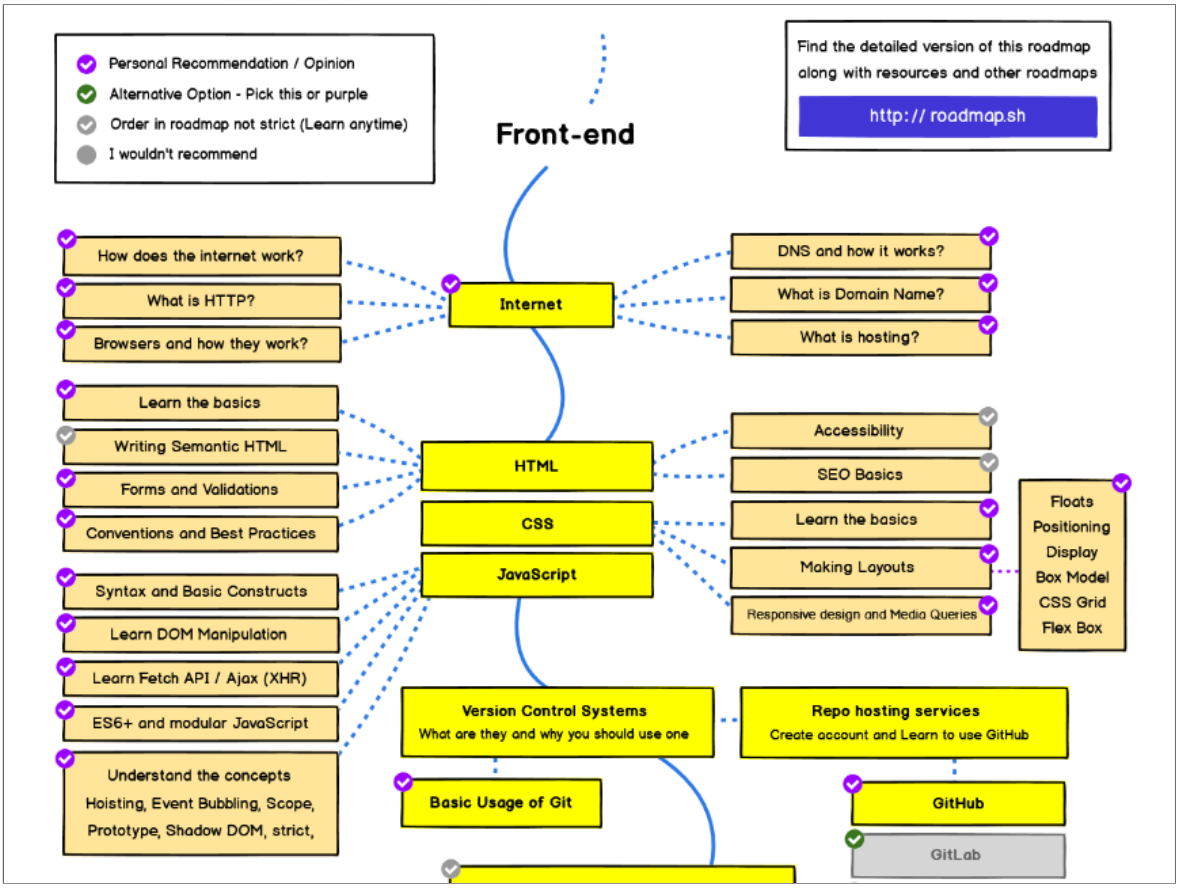
Cursada 2020

TEMARIO

- JS
- DOM
- AJAX
- CORS

ARRANQUEMOS CON LAS PREGUNTAS DE REPASO

PROCESAMIENTO EN EL CLIENTE



¿QUÉ SABEMOS DE LOS SCRIPTS?

- Los scripts son pequeños programas que se incluyen en el documento html.
- Si el navegador NO puede ejecutar scripts, existen la etiqueta **noscript**:

```
 <script>
alert("Hola!")
</script>
<noscript>Esta navegador no suporta JavaScript!</noscript>
```

- Pueden estar incluidos directamente en el documento html o en archivos externos.

```
<html> <head>
<script  src="misScripts.js">
</script>
</head>
<body>
<script>
...código....
</script>
</body></html>
```

¿SIEMPRE JAVASCRIPT?

- El lenguaje Javascript por defecto (a partir de HTML5.2 quedó **obsoleto atributo language**, por ejemplo.
- Es posible escribir scripts en otros lenguajes.
- Se usa el atributo **type** para indicar el tipo de script.

```
<script type="application/ecmascript"...></script>
<script type="application/javascript"...></script>
<script type="text/javascript"...></script>
```

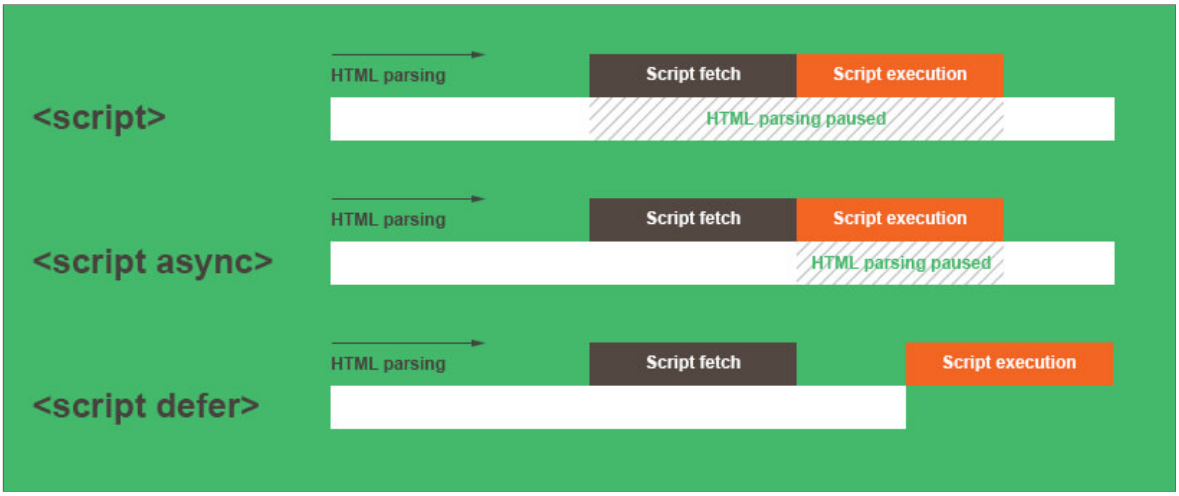
Definidos por **IANA Media Types**.

JAVASCRIPT: CARACTERÍSTICAS BÁSICAS

- Dialecto del estándar **EcmaScript**. El estándar es el **ECMA 262**
- JavaScript es un lenguaje de programación interpretado.
- Puede correr tanto en el cliente como en el servidor.
- **NOSOTROS lo usaremos en forma client-side**: ¿dónde está el intérprete?

FORMA DE EJECUCIÓN

- Atributos async y defer:



Más info en: **https://www.w3.org/TR/html52/semantics-scripting.html#element-attrdef-script-defer**

LA CONSOLA

- Probemos...

```
    parseInt("1234");
    parseInt("11",2);
    parseInt("0x10");
    parseInt("hola");
    Math.sqrt(9);
    Number.MAX_VALUE;

    var  x=  false;
    if (x) alert("hola");
    "proyecto".length;
    "proyecto".charAt(2);
    "proyecto de desarrollo".replace("desarrollo",
"software");
    "proyecto".toUpperCase();

    var arreglo = new Array("uno", 2, "tres");
    arreglo[3]="3333";
    var frutas= new Array();
    frutas["citricos"]=new Array("naranja", "pomelo");
    frutas["otros"]=new Array("manzana", "pera");
    frutas["citricos"];
    frutas.citricos;
```

SEGUIMOS PROBANDO...

```
    hoy = new Date();
    findel2020 = new Date(2020, 11, 31);

    dias = new
Array("dom","lun","mar","mier","jue","vie","sab");
    alert(dias[hoy.getDay()]);
```

- Más ejemplos en este **tutorial básico**

HABLAMOS DE SCRIPTS, PERO...

¿Para qué los utilizamos en nuestros desarrollos?
Si queremos cambiar el contenido dinámicamente de mi página, o hacer validaciones, entre otras muchas cosas ¿qué necesitamos?

DOM: EL MODELO DE OBJETOS DEL DOCUMENTO

DOM: EL MODELO DE OBJETOS DEL DOCUMENTO

- El modelo de objetos del documento es una **API**, que permite acceder a los contenidos de un documento HTML/XML.
- Proporciona una **representación** estructurada, **orientada a objetos**, de los elementos individuales y el contenido del documento, con métodos para recuperar y fijar las propiedades de dichos objetos.
- Proporciona métodos para agregar y eliminar objetos al documento.
- También proporciona una **interfaz estándar** para trabajar con **eventos**.
- La especificación en: **https://dom.spec.whatwg.org/**

EL DOCUMENTO SE VE COMO UN ÁRBOL DE NODOS

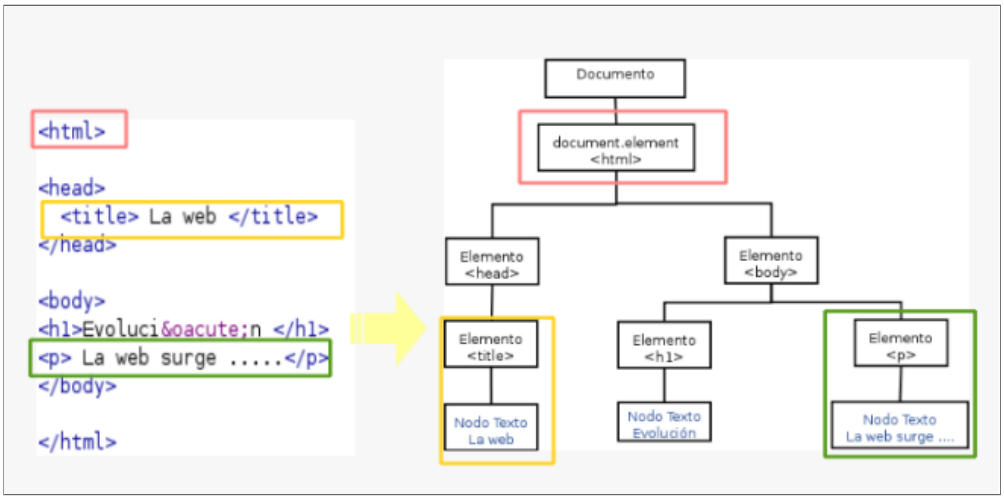
- Cada nodo tiene sus propios métodos y propiedades, pero todos implementan la **interfaz Node**: un conjunto común de métodos y propiedades relacionadas con la estructura de árbol del documento.
- Por ejemplo: **insertBefore()**, **appendChild()**, **removeChild()**, entre otras.
- Por ejemplo: **firstChild**, **lastChild**, **childNodes**, **parentNode**, entre otras.

EN UN DOCUMENTO HTML

- El documento entero es un **nodo documento**.
- Cada elemento HTML es un **nodo elemento**.
- Los textos que aparecen en las páginas son **nodos de texto**.

EL DOCUMENTO

- La raíz del árbol es el **objeto document**, que implementa la **interfaz Document**.
- El objeto document tiene sólo un elemento hijo, dado por **document.documentElement**.
- **document.documentElement** corresponde al elemento **<html>** y tiene dos hijos: **<head>** y **<body>**



INTERFAZ DOCUMENT

- La **interfaz Document** proporciona métodos para acceder y crear otros nodos en el árbol del documento tales como:
    - **getElementById()**
    - **getElementsByTagName()**
    - **createElement()**
    - entre otros...

DOM – ALGUNAS HERRAMIENTAS

RECORRIENDO EL ÁRBOL

Miremos el siguiente código:

```
<html>
    <head>
        <title></title>
    </head>
    <body>
        <h1>Algo.......</h1>
        <p>bla bla bla </p>
    </body>
</html>
```

- ¿Qué resultado dá lo siguiente: **document.documentElement.lastChild.firstChild.tagName**?

ACCEDIENDO A LOS NODOS

- A través del atributo **id**, podemos utilizar el método: **document.getElementById()**
- Para recuperar todos los elementos de un mismo tipo, se puede usar el método: **document.getElementsByTagName()**

```
<script>
  alert(document.getElementById("p1").nodeType);
  var x = document.getElementsByTagName("p");
  for (var i = 0; i<x.length; i++){
    alert(x[i].innerHTML);
  }
  var x = document.getElementsByTagName("a");
  for (var i = 0; i<x.length; i++){
    x[i].style.color = "red";
  }
</script>
```

ACCEDIENDO A NODOS: QUERYSELECTOR Y QUERYSELECTORALL

```
<script>
    var primer_parrafo = document.querySelector("p.especial");
    primer_parrafo.style.color = "red";
</script>
```

```
<script>
    var x = document.querySelectorAll("p.especial");
    for (var i = 0; i<x.length; i++)
    {
      x[i].style.color = "red";
  }
</script>
```

MUCHAS LIBRERÍAS JS BRINDAN ATAJOS

- Atajos a las funciones de DOM en JQuery:
  - **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>**

MODIFICANDO EL ÁRBOL

- La interfaz document incluye métodos que permiten modificar el árbol de nodos: **createElement**, **createTextNode**
- **Ejemplo**: quiero crear una lista en forma dinámica....

```
...
var lista=document.createElement("ul");
var item=document.createElement("li");

....
lista.appendChild(item);
....
document.documentElement.lastChild.appendChild(lista);
....
```

- Veamos **este ejemplo de listas**

TIPOS DE NODOS

- **Nodos elementos**: corresponden a las etiquetas del documento. Pueden tener nodos hijos, que pueden ser otros elementos o nodos de texto.
- **Nodos de texto**: representan contenido, o simplemente caracteres. Tienen un nodo padre y, posiblemente, nodos del mismo nivel, pero no pueden tener nodos hijos.
- En realidad hay más...

NODOS DE TEXTO

- Los nodos de texto no tienen un atributo **id**.
- No se pueden acceder mediante los métodos **getElementById()** o **getElementsByTagName()**.
- **Se acceden a través de su nodo padre.**
- Ejemplo:

```
.....
<p id='p1'> Texto inicial ....</p>
<script>
....
document.getElementById('p1').firstChild.nodeValue='Otro';
</script>
```

- Veamos **este ejemplo**
- nodeValue vs innerHTML.

DOM Y EVENTOS

DOM también contempla...

... un **sistema de eventos** genérico que permita registrar **manejadores** de eventos, describir el **flujo de eventos** a través de la estructura del árbol y proveer **información contextual** sobre cada evento.

- También define un subconjunto común de los sistemas de eventos actuales.

MODELO DE EVENTOS

- ¿Qué es un evento? ¿Cuándo se produce?
- La especificación de DOM:
    - Define y explica la propagación y registro de eventos.
    - Define la **interfaz Event**.
    - Define cómo se interpreta el flujo de eventos una vez producido.

FLUJO DE EVENTOS

Imagen obtenida de **w3.org**

MANEJADORES DE EVENTOS

```
function manejador(evento) {
    //
    // "evento" se crea implícitamente y contiene la
    // info sobre  el evento producido.
    //
    }
    var elem=document.getElementById('p1');
    elem.onmouseover = manejador;
```

manejador es un objeto function.

ESCUCHADORES DE EVENTOS

- Los objetos DOM también pueden ser registrados como **escuchadores de eventos**.
- Esta característica puede ser utilizada para asignar múltiples manejadores para un evento dado.
- Los métodos básicos son:**addEventListener** y **removeEventListener**

ESCUCHADORES DE EVENTOS: EJEMPLO

```
var elem = document.getElementById('p1');
elem.addEventListener("mouseover", f1, true):
elem.addEventListener("mouseout", f2, true):
.......

elem.removeEventListener("mouseover", f1, true);


elem.addEventListener("mouseover", f1, true):
elem.addEventListener("mouseover", f1, false):
```

UN POCO MÁS DE JAVASCRIPT

EXCEPCIONES EN JAVASCRIPT

- Se pueden lanzar excepciones usando la sentencia **throw**.
- Se manejan con la sentencia **try...catch**.
- Un **ejemplo de la w3schools**
- **finally** opcional.
- Más info **developer.mozilla**

OBJETOS Y JAVASCRIPT

```
var mi_cancion = {
        "titulo": "Ruta 66",
        "interprete": "Pappo",
        }
```

```
    function Musica(titulo, interprete){
        this.titulo=titulo;
        this.interprete=interprete;
            }
    var mi_musica= new Array();
    mi_musica[0]= new Musica(
        "Desconfio",
        "Pappo");
    mi_musica[1]= new Musica(
        "Sussy Cadillac",
        "Riff");
    mi_musica[2]= new Musica(
        "Llegara la paz",
        "Pappo's Blues");
```

- ¿Qué representa la función Musica?
- ¿**this**?

OBJETOS Y JAVASCRIPT

Ahora con class

```
class Musica{
 constructor (titulo, interprete){
    this.titulo=titulo;
    this.interprete=interprete;
        }
}
```

- ¿Qué diferencias hay con la función Musica?
- Más info en este **artículo sobre modelo de objetos.**
- Y en la **definición del estándar**

OBJETOS Y JAVASCRIPT

Y surgió JSON ...

- **JSON**: JavaScript Object Notation.
- Surge como una forma de definir objetos en Javascript.

```
var mi_cancion = {
    "titulo": "Lily Malone",
    "interprete": "Riff",
    }
```

- Es un formato ligero para el envío y recepción de datos.

JSON

- JSON se basa en dos estructuras:
  - Objetos: una colección de pares de nombre/valor
  - Arreglos: una lista ordenada de objetos

```
pappo1 = {
    "titulo": "Blues para mi guitarra",
    "interprete": "Pappo",
    }
pappo2 = {
    "titulo": "El hombre suburbano",
    "interprete": "Pappo",
    }
coleccion= [pappo1, pappo2];
```

```
var mi_musica = [
    {titulo: "Desconfio",
    interprete: "Pappo"
    },
    {titulo: "Sussy Cadillac",
    interprete: "Riff"
    },
    {titulo: "Llegará la paz",
    interprete: "Pappo's Blues"
    }
    ];
```

MUY USADO

- Veamos este ejemplo: **https://developers.mercadolibre.com.ar/es_ar/categorias-y-publicaciones#close**
- ¿Conocen la **central meteorológica** de la facultad? Miremos estos **datos**
- Hay alternativas como XML y YAML.

AJAX: ASYNCHRONOUS JAVASCRIPT + XML

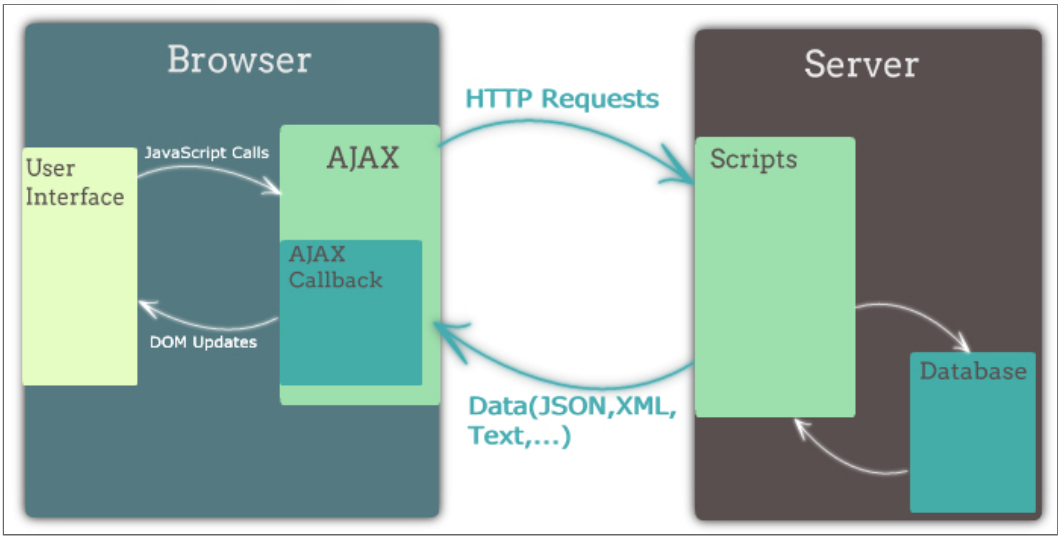
AJAX

- 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):

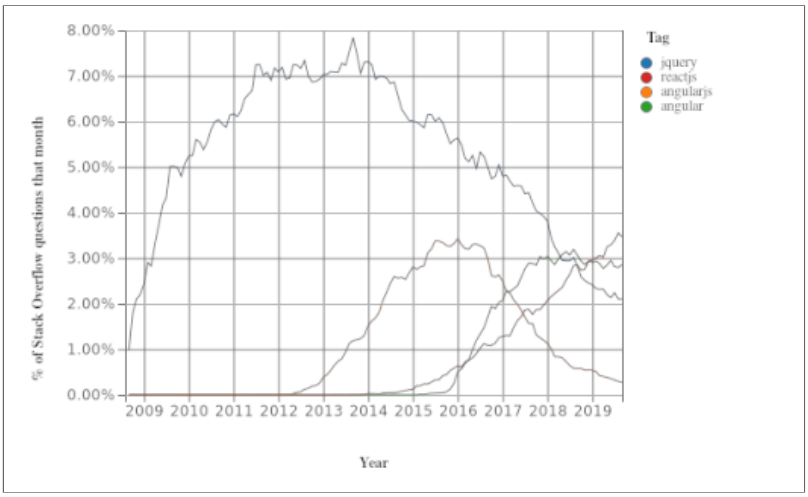
```
    function buscarSync() {
      xhr = new XMLHttpRequest();
      var param = document.getElementById('interprete').value;
      var url = "http://localhost:5000/musicos?name=" +
escape(param);
      xhr.open("GET", url , false);
      xhr.send();

      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

- En **javascripting.com** hay una lista de más de 1000 registradas.
- 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**

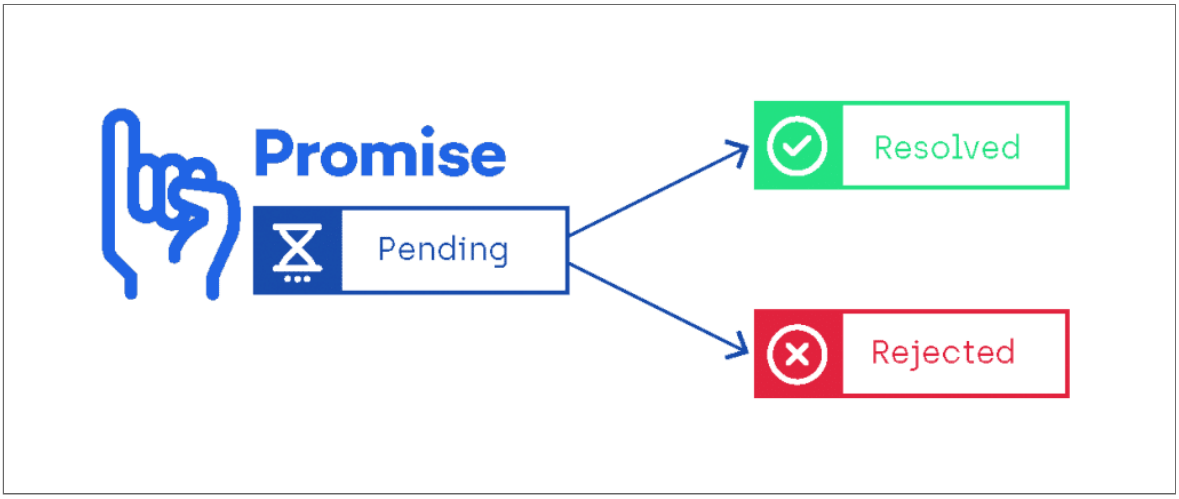
```
$.ajax({
  url: '/ruta/hasta/pagina',
  type: 'POST',
  async: true,
  data: 'parametro1=valor1&parametro2=valor2',
  success: procesaRespuesta,
  error: muestraError
});
```

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**.

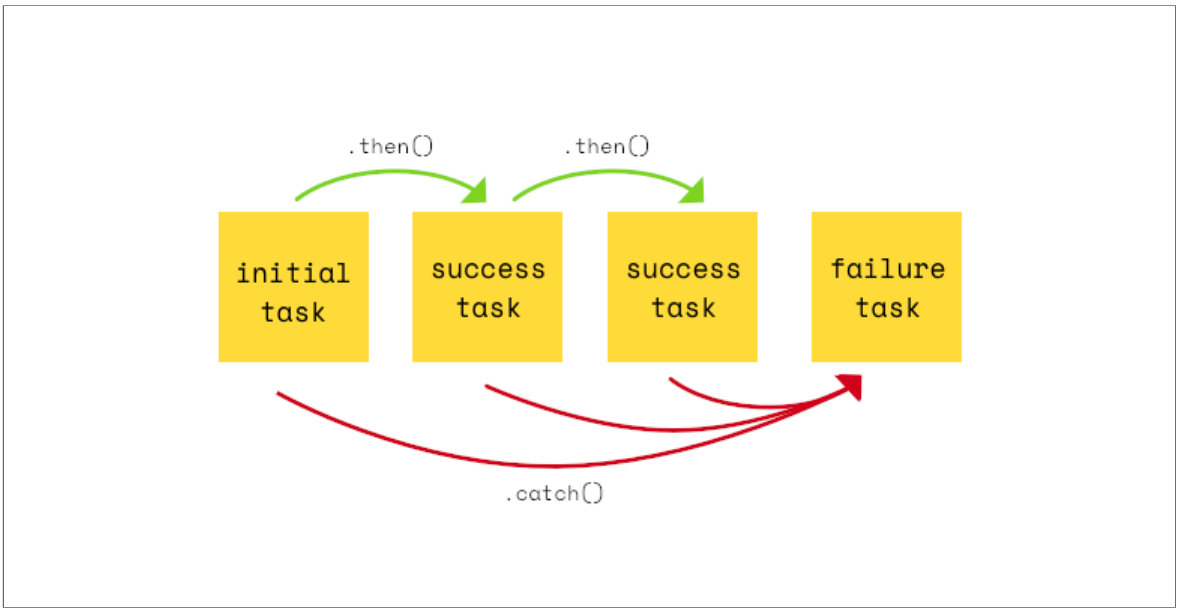
PROMESAS JS

*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 getUserAsync(name)
  {
    let response = await
fetch(`https://api.github.com/users/${name}`);
    let data = await response.json()
    document.getElementById('info').textContent =
JSON.stringify(data);
  }
```

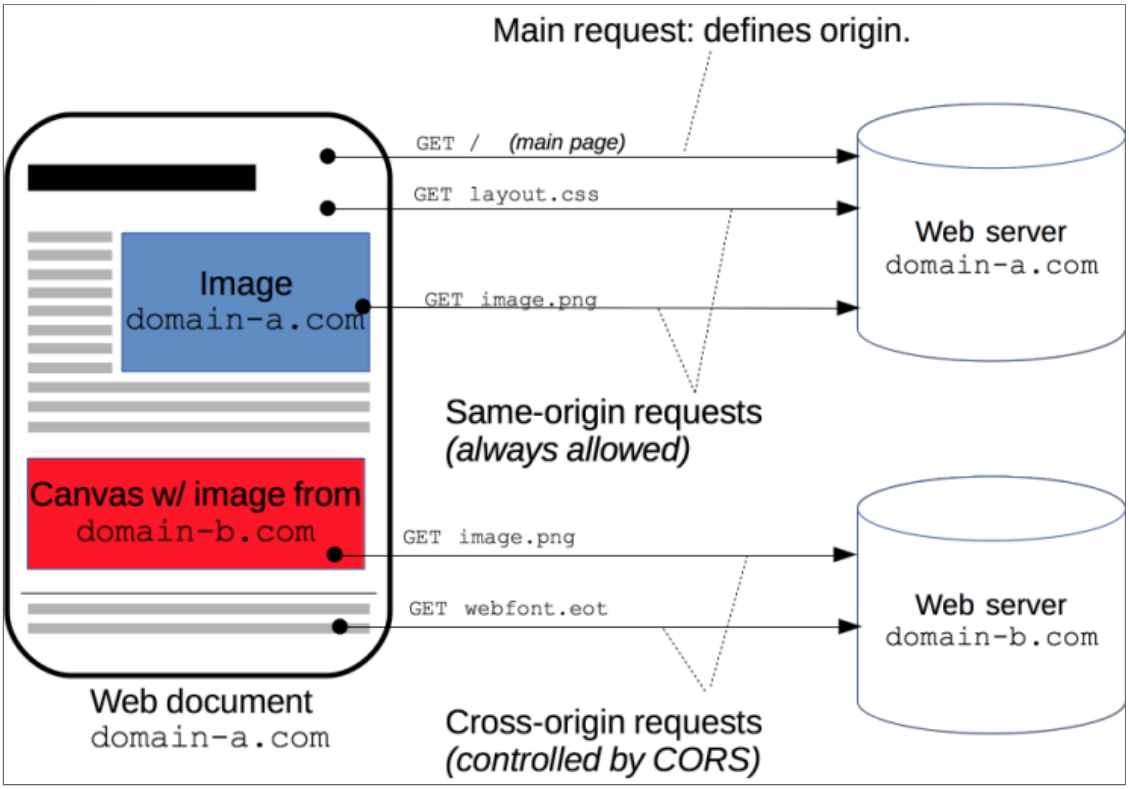
- Referencia **async** y **await**.

¿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
Image
domain-a.com
Canvas w/ image from
domain-b.com
GET image.png
GET webfont.eot
Web server
domain-a.com
Web server
domain-b.com
Same-origin requests
(always allowed)
Cross-origin requests
(controlled by CORS)
Web document
domain-a.com

EL CASO MÁS SIMPLE

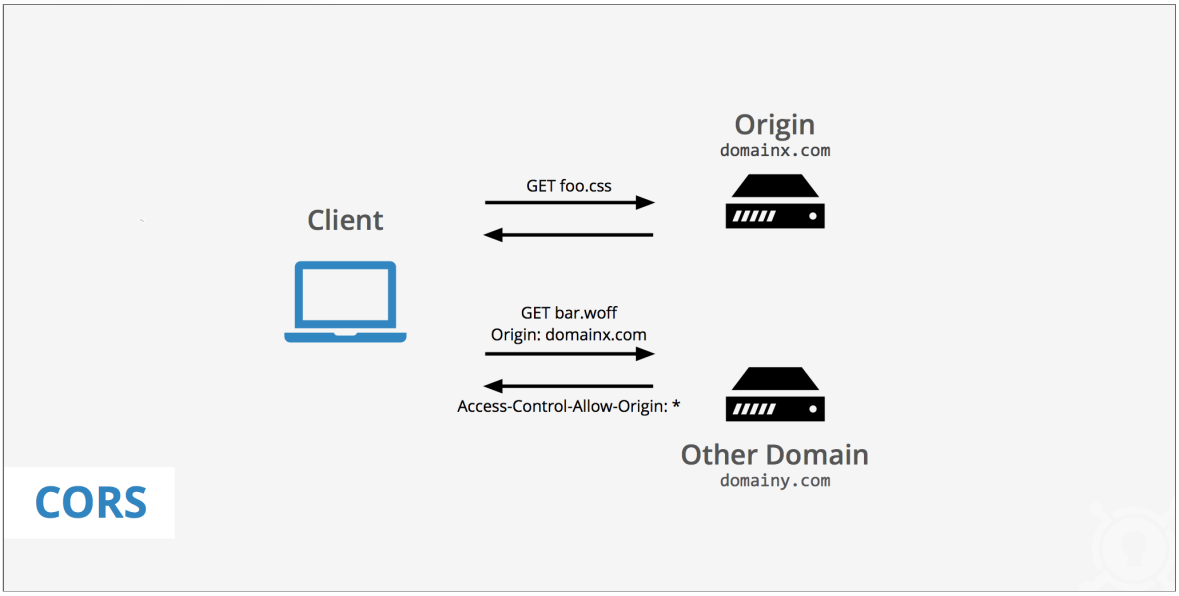
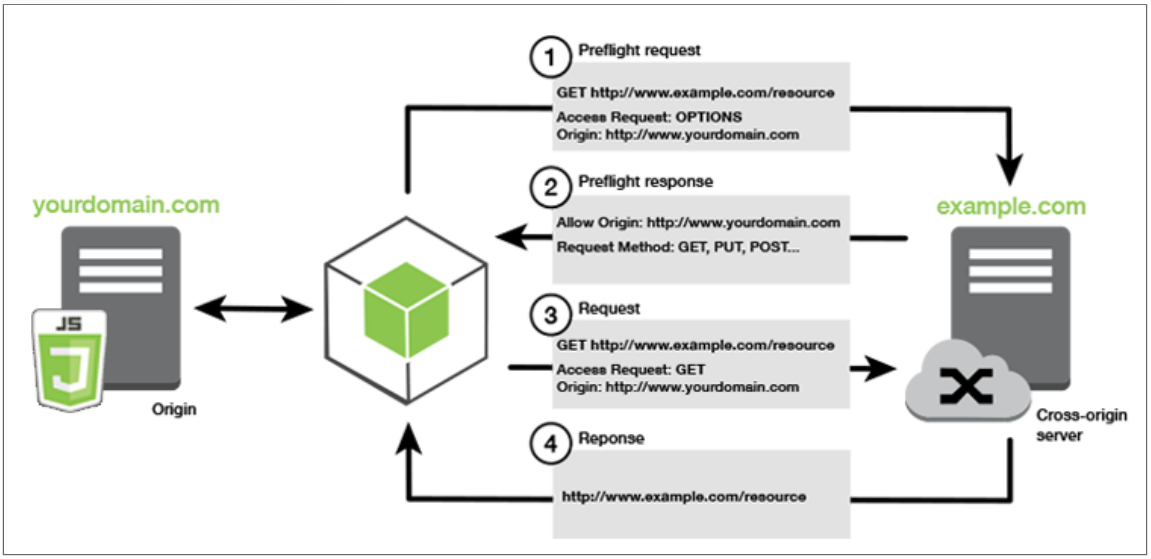


DIAGRAMA COMPLETO



REFERENCIAS CORS

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

¿DUDAS?

¿CUÁNTAS HORAS ESTUVE HABLANDO?

SEGUIMOS LA PRÓXIMA MEJOR...