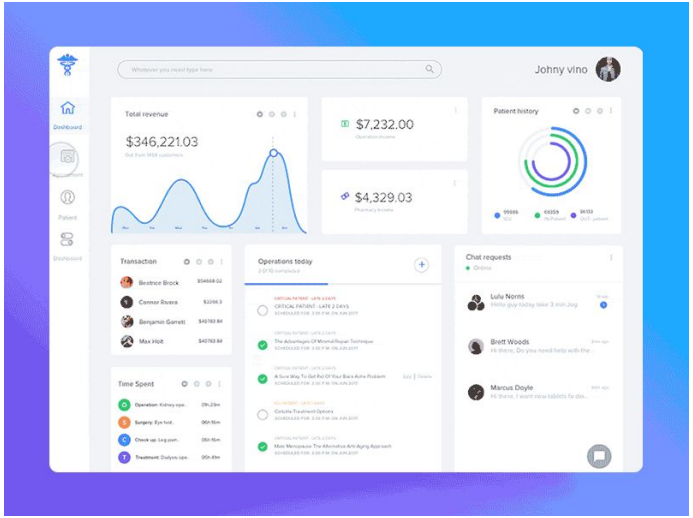
SPA y AJAX

Single Page Application (SPA)

- Es un tipo de aplicación web
- Nunca se recarga completamente el navegador
- Los recursos se cargan parcial y dinámicamente cuando lo requiera la página



Introducción

AJAX

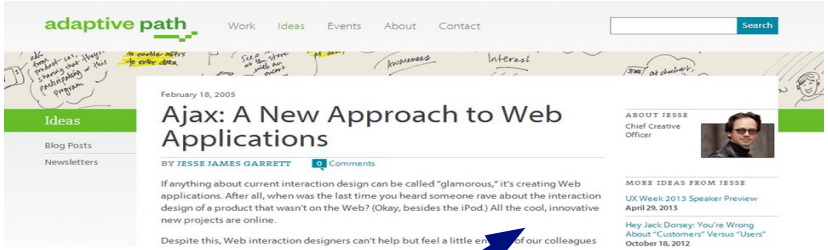
Asynchronous Javascript And XML.

Técnica frontend que es **base** de las **aplicaciones web modernas**.

- No es un lenguaje o tecnología.
- Es una técnica que **combina un set de tecnologías** conocidas, para hacer mejorar la experiencia de usuario en las páginas web (más amigables y rápidas)

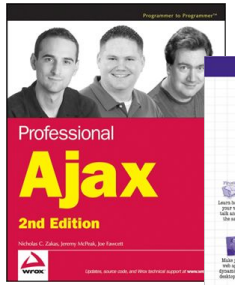
Evolución: Un poco de historia

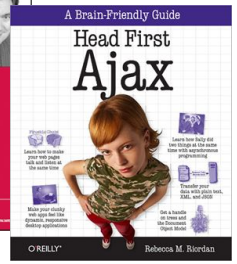
Link: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/

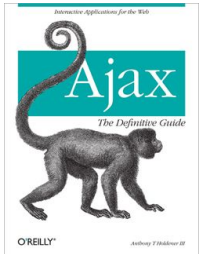


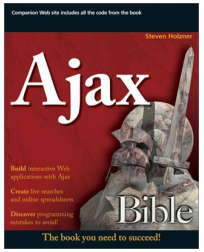
ENTRA EN DESUSO LA PALABRA AJAX

Surge el nombre y concepto → Boom en la industria → SPA en todos lados



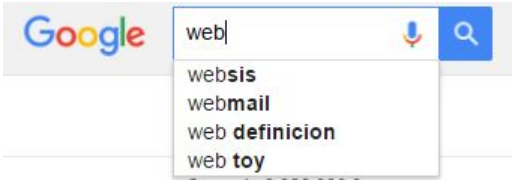




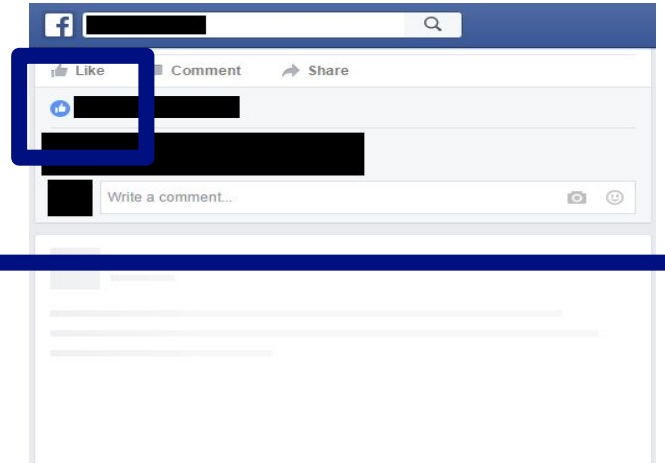


Ejemplos

Sugerencias sin recargar la página.



Likes

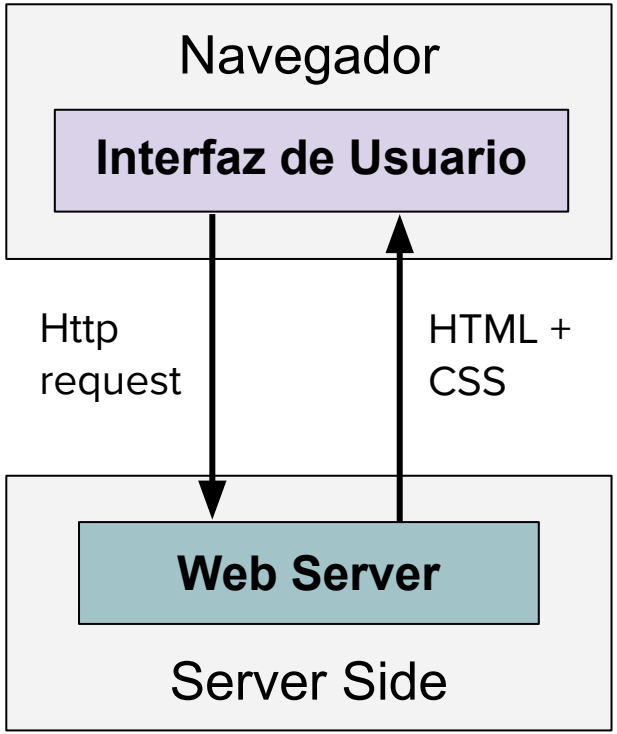


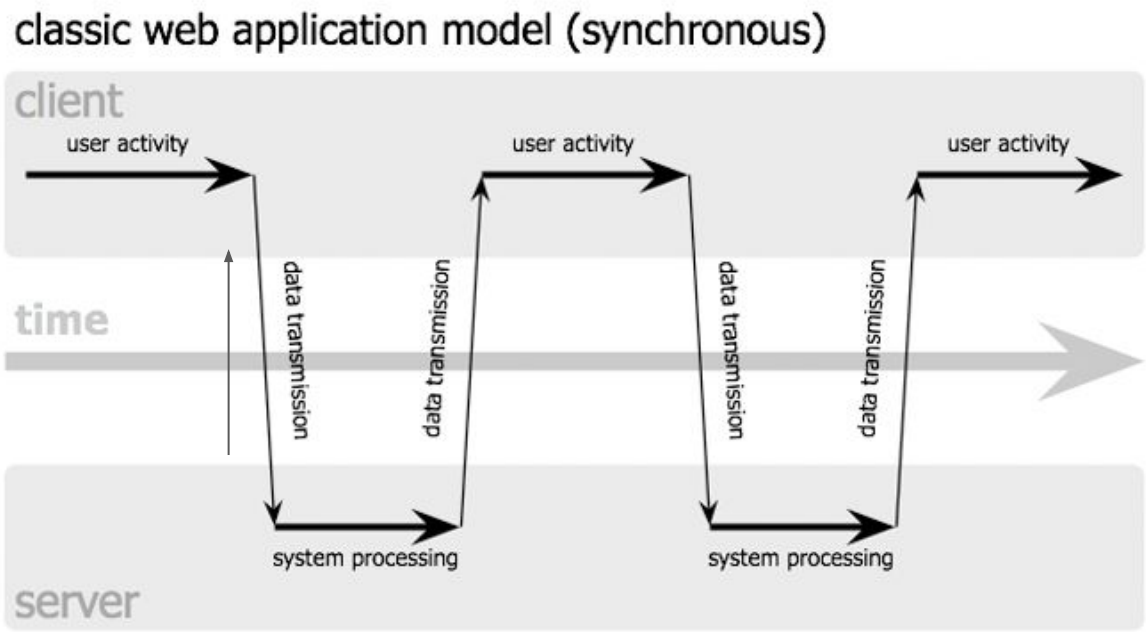
Carga asincrónica

Edición “en línea” y guardado automático.

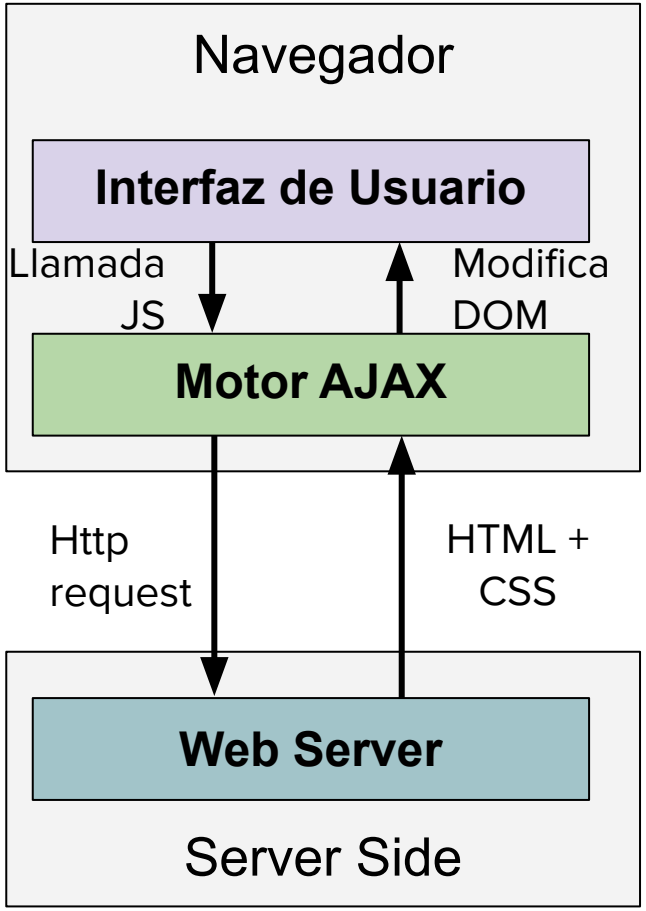

gifak.net

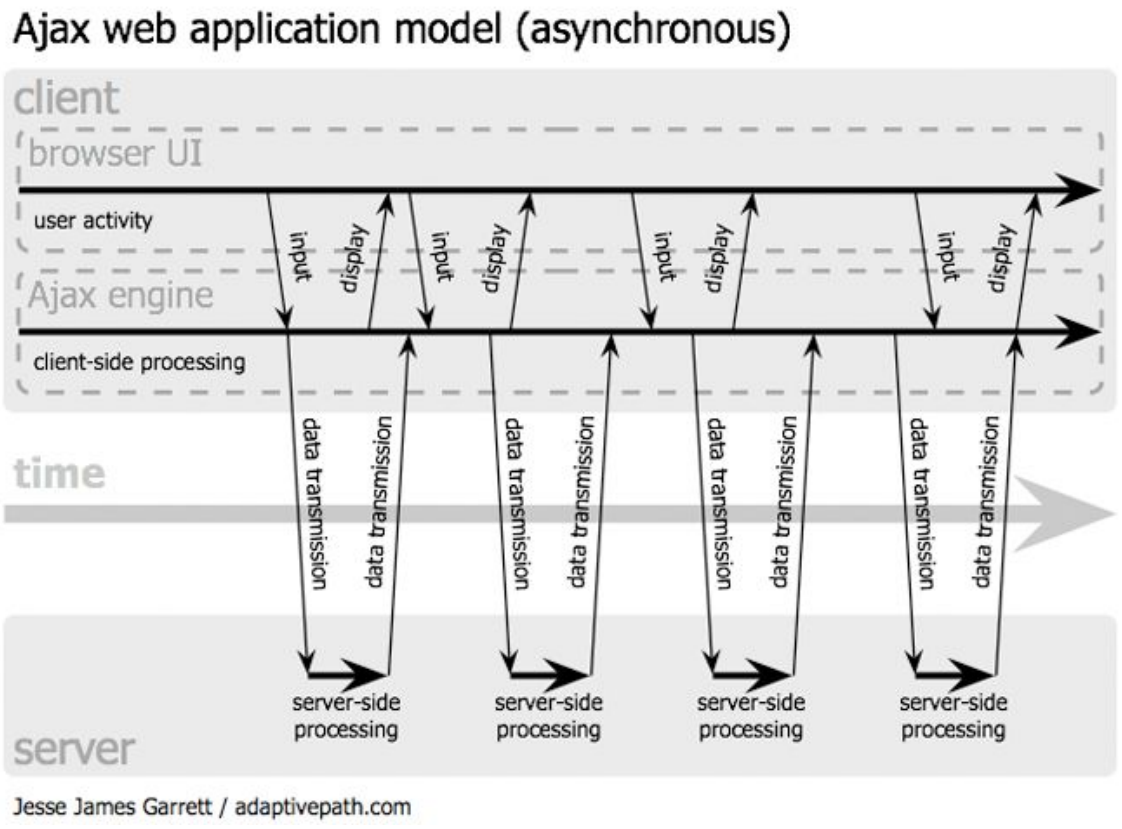
Una web app sin AJAX



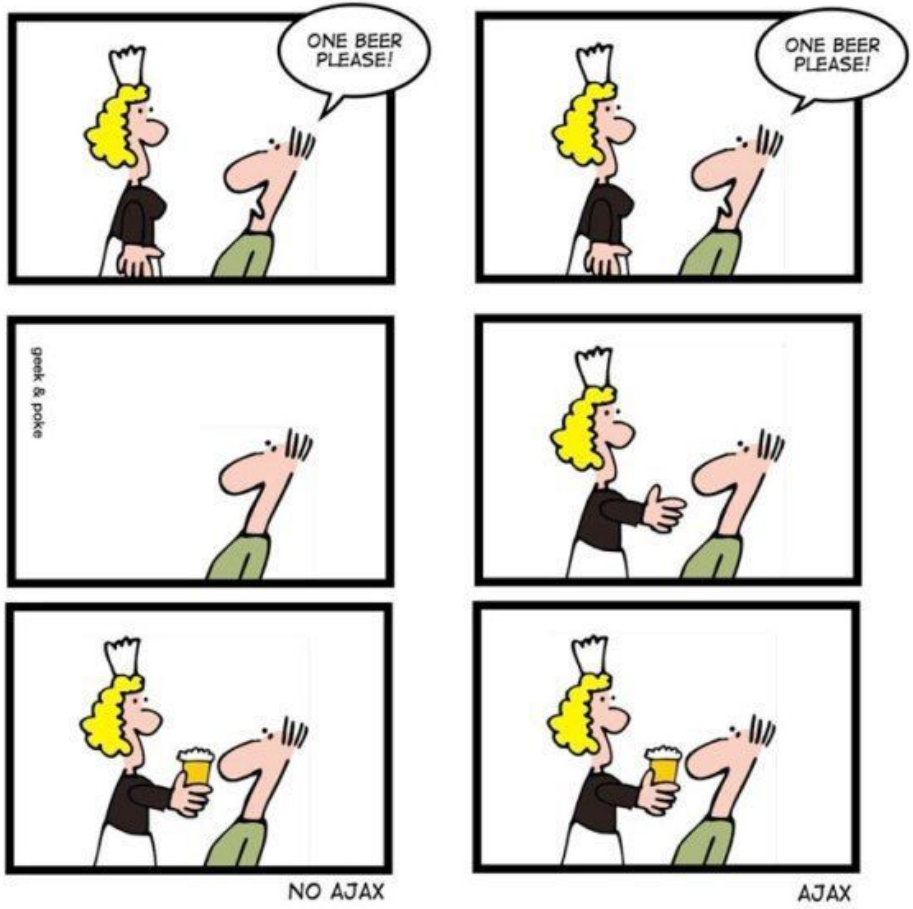


Una web app con AJAX





SIMPLY EXPLAINED



Entonces, ¿Qué es AJAX?

- Técnica de carga asíncrona de contenido dinámico y datos del server.
- Permite **cambios dinámicos** del lado del cliente.
- Mejora la **experiencia del usuario**.

Hasta ahora teníamos:

- **DOM (HTML)**
- **Estilos (CSS)**
- **Programar y editar el DOM (JS)**
  - Todo dentro del cliente

Y ahora!!!

- **Conexión al servidor!**
  - Asincrónica (es JS)

Ventajas / Desventajas

- Mejora experiencia de usuario.
- Majora velocidad.
- Disminuye el volumen de datos que se transfieren

- Puede fallar, y agrega complejidad.

Ventajas / Desventajas

- Sede Tres Arroyos [TBC]
- Velocidad Respuesta
- No hay que recargar página, menos volumen de información para transmitir
- Se notan errores de comunicación
- Se puede seguir interactuando con la página

Desventajas

Complejidad y curva de aprendizaje, debug

Ventajas / Desventajas TANDIL 2020

- [TBC]
+ Pagina mas dinamica
+ Mas velocidad
+ El user no pierde tiempo
+ Guardar datos en el server

- Carga optimista puede fallar y confundir al usuario
- Sigue dependiendo de la conexion
- Necesita aprendizaje nuevo

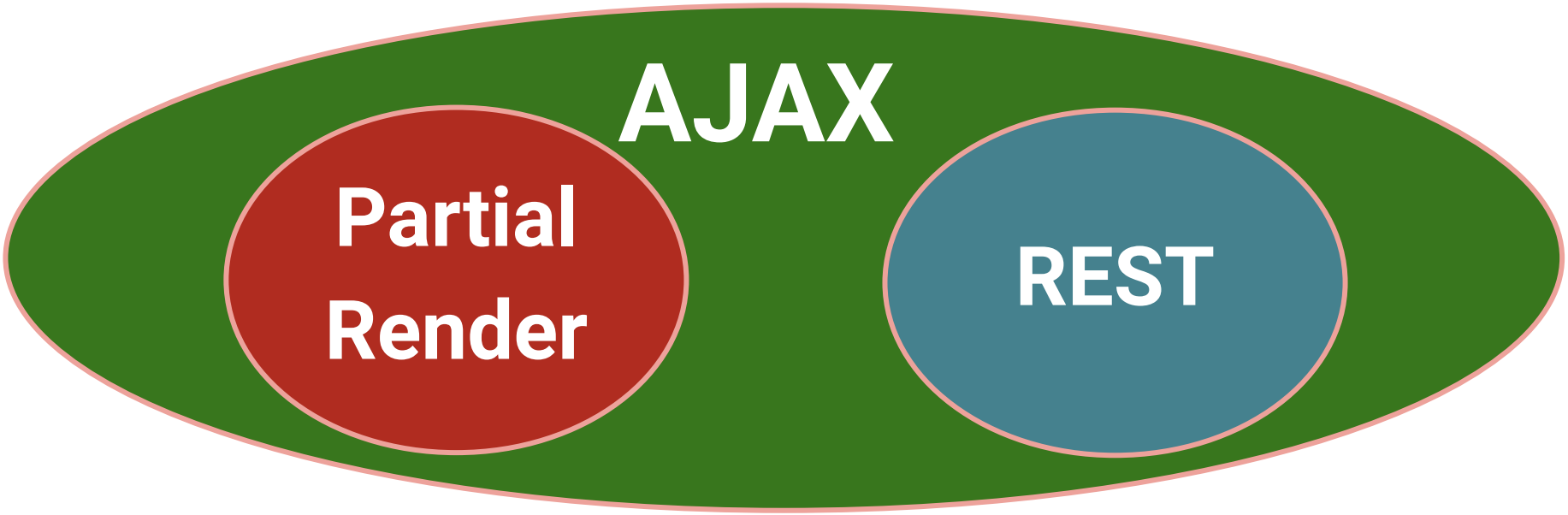
Ventajas / Desventajas TANDIL 2022

- No hace recargar todo el contenido, la pagina se vuelve mas dinamica
- Mas facil escalar porque solo se agrega parte del contenido
- La peticion puede fallar desventaja o ventaja de acuerdo como se mire.
- Mas amigable con el usuario y no confunde con la carga pendiente.
- Uso de Contenido y datos de otras paginas/aplicaciones

Desventaja:

- Sensible a conexion.
- Curva aprendizaje y capacitación
- Mayor complejidad en desarrollo

Estilos de AJAX



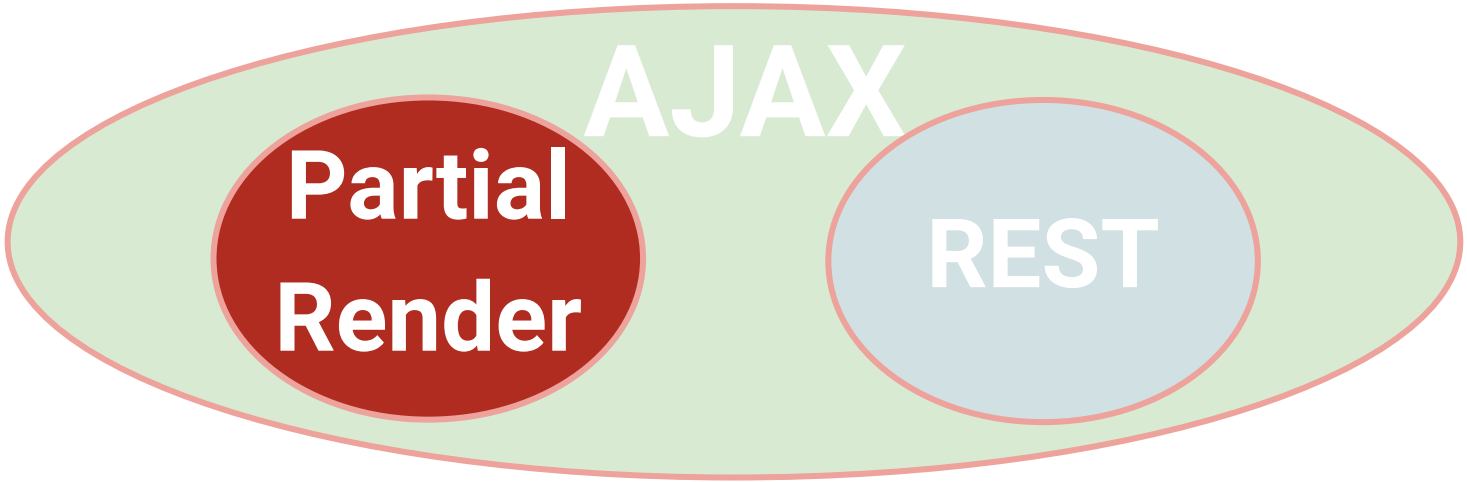
- **Partial render de páginas**
  - Carga un fragmento HTML y lo inserta en nuestro html
- **Servicio REST**
  - Consulta un objeto JSON y lo procesa del lado del cliente con Javascript


THIS IS
AJAX
imgflip.com

Partial Render con AJAX

Estilos de AJAX

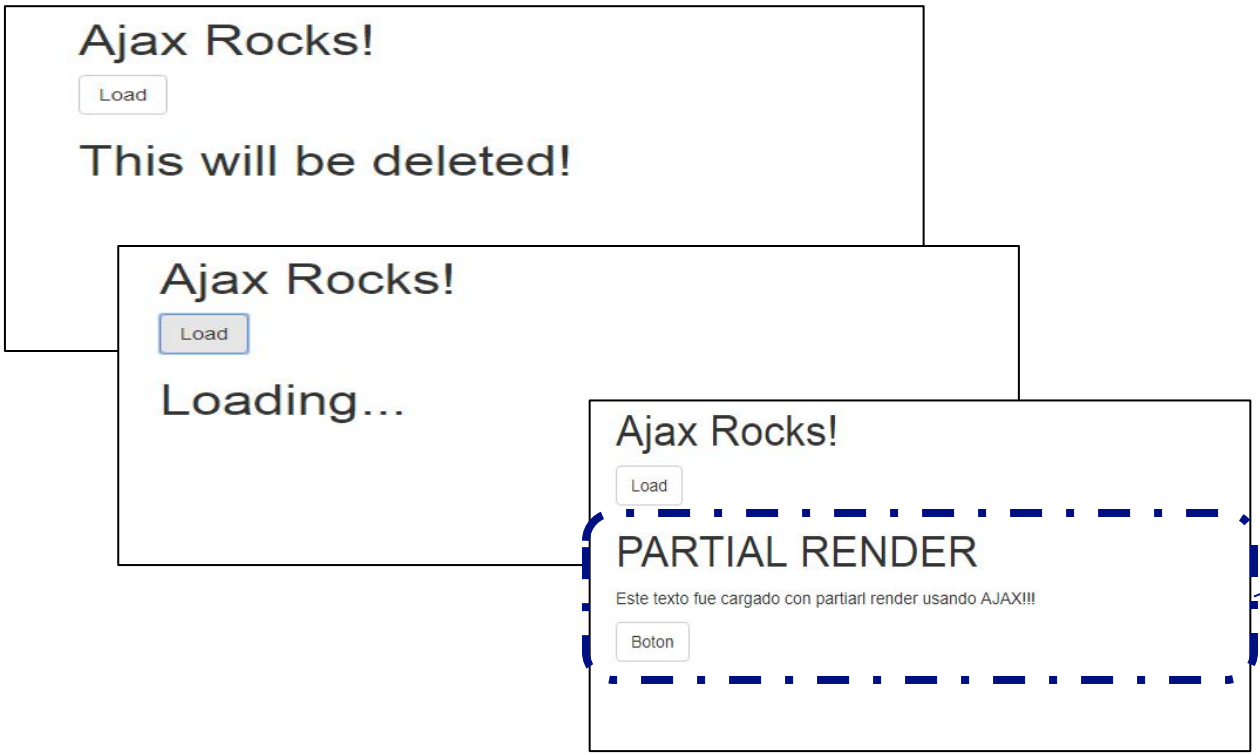
- **Partial render de páginas**
  - Carga un fragmento HTML y lo inserta en nuestro html
- **Servicio REST**
  - Consulta un objeto JSON y lo procesa del lado del cliente con Javascript



Ejemplo - Partial Render

- Vamos a apretar un botón y mostrar un pedacito de HTML dentro de un DIV.
- El contenido lo cargaremos desde:
  - http://web-unicen.herokuapp.com/api/html?
- Localmente puede decargar los ejemplos de https://drive.google.com/drive/folders/1EVePi-Ss4p41T-BM_LqiP2dHq4QH5fRe?usp=sharing

Ejemplo - Partial Render



Esto es un fragmento de HTML que bajamos desde esa URL o descargamos

Pasos

¿Cómo mostramos ese fragmento dentro del sitio usando **partial render?**

- Crear HTML
- Agregar un evento en el botón
- Bajar el archivo (hacer un pedido HTTP al servidor)
- Mostrar el archivo dentro del

¿Qué partes sabemos hacer?

¿Cuáles no?

Ajax en ES7

ES7 incorpora la interfaz **fetch()** para llamados **ajax**

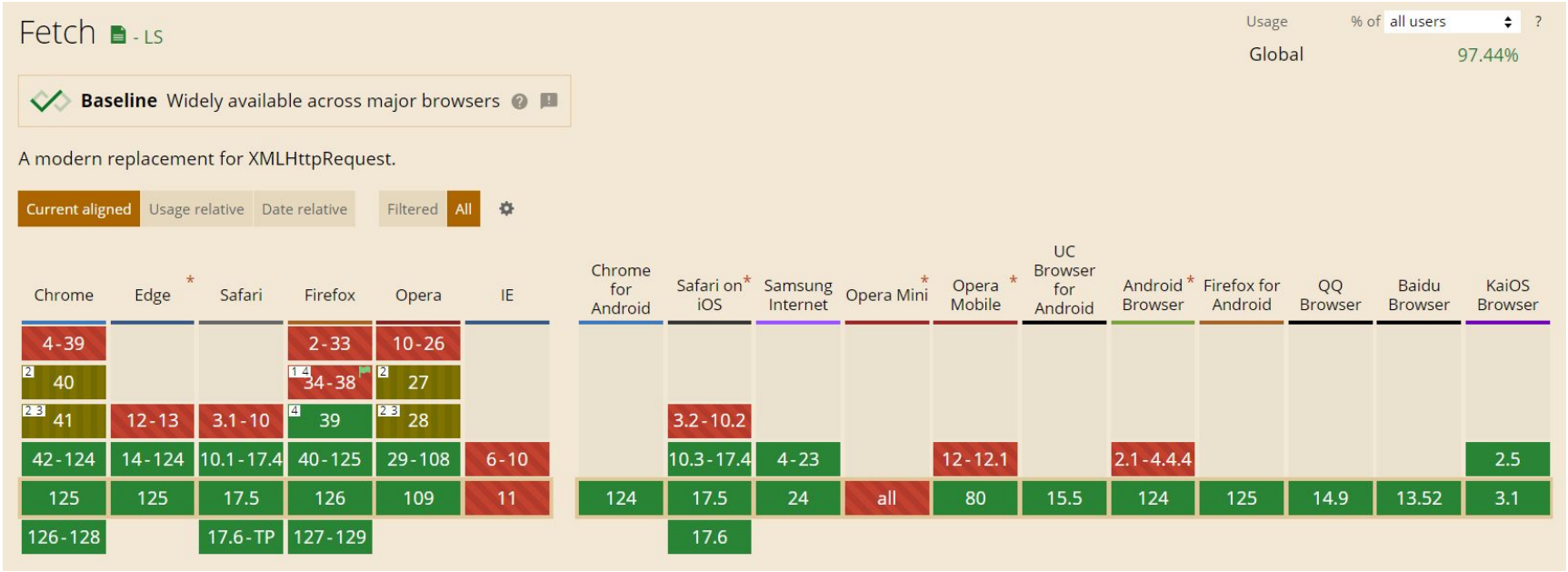
```
let promise = fetch(url);
promise.then(response => ...do something )
```

O la versión corta

```
fetch(url).then(response => ...do something )
```

El uso más simple de fetch() toma un argumento (la ruta del recurso que se quiera traer) y **el resultado es una promesa** que contiene la respuesta (un objeto Response)

Ajax en ES7 - fetch



https://caniuse.com/#feat=fetch

https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch

Promesas

JavaScript es de un solo hilo (mono thread), es decir, dos porciones de secuencia de comandos no se pueden ejecutar al mismo tiempo. Solamente puede ejecutar uno después del otro.

- ES6 introduce ***promises*** (promesas en castellano)
- Son un objeto que representa la terminación o el fracaso de una operación asíncrona.
  - Es la forma de implementar asincronismo en JS

Una paquete que adentro va a tener el resultado asincrónico (en el futuro). Cuando llega lo tenes que “desenvolver”.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Let's code it

La carga parcial consiste entonces en ir a buscar texto HTML a un archivo externo y ponerlo en un div con ID "use-ajax"

```
fetch("otro-archivo.html").then(response => {
    response.text().then(texto => {
        document.querySelector("#use-ajax").innerHTML = texto;
    });
});
```

- Como sabemos que la respuesta es **texto HTML**, tenemos que procesarla usando `response.text()`
- **El procesamiento de la respuesta es otra promesa!**

Ejemplo - Partial Render

HTML

```html
<h1>Ajax Rocks!</h1>
<button class="btn btn-default js-load">Load</button>
<div id="use-ajax">
    <h1>This will be deleted!</h1>
</div>
```

JS

Ejemplo1 - https://codepen.io/webUnicen/pen/rKNyPK

```js
function loadClick(event) {
    event.preventDefault();
    fetch("https://web-unicen.herokuapp.com/api/html?")
        .then(response => {
            response.text().then(text => {
                document.querySelector("#use-ajax").innerHTML = text
            });
        });
}
let jsloads = document.querySelectorAll(".js-load");
jsloads.forEach(e => e.addEventListener("click", loadClick));
```

Anduvo!

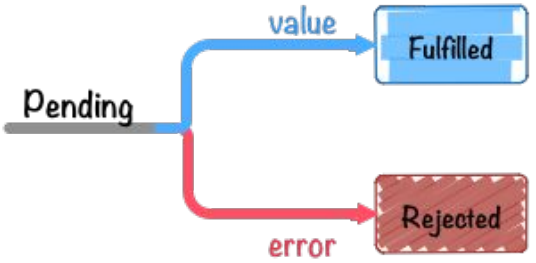
Y qué pasa si la descarga no se puede hacer?

Falsas promesas

Promesas (Terminología)

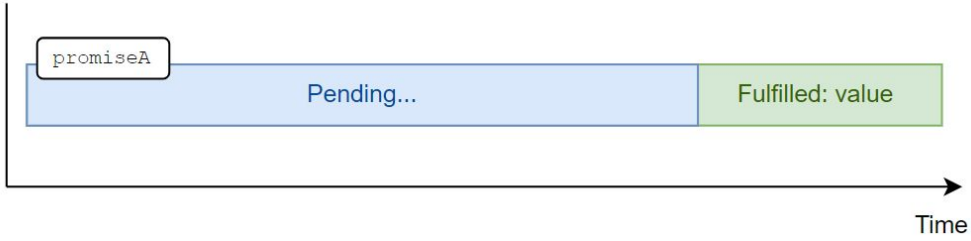
Una promesa tiene 4 estados

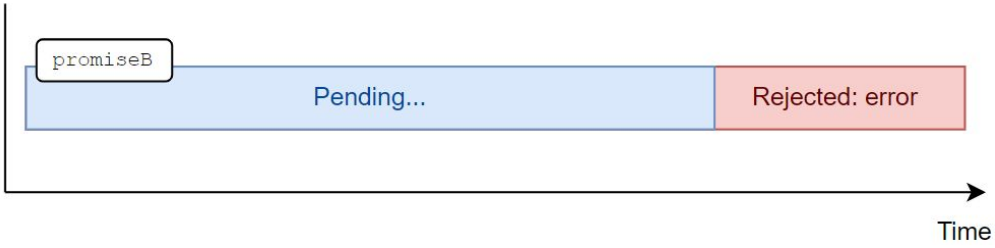
- Cumplida *(fulfilled)*
- Rechazada *(rejected)*
- Pendiente *(pending)*
- Finalizada *(settled)*



promise

```
hazAlgo()
    .then(exitoCallback)
    .catch(falloCallback);
```





Ejemplo - Partial Render

```
function loadClick(event) {
    event.preventDefault();
    fetch("https://web-unicen.herokuapp.com/api/html?")
        .then(response => {
            response.text().then(text => {
                document.querySelector("#use-ajax").innerHTML = text
            });
        })
        .catch(error => {
            console.log(error);
            container.innerHTML = "<h1>Error - Connection Failed!</h1>";
        });
}
```

Ejemplo 2 (apagar local server) - https://codepen.io/webUnicen/pen/qKBrMg

¿Y si la conexión anduvo pero el archivo no existe?

response

El objeto “response” tiene información de la respuesta obtenida del servidor.

Con response.ok nos dice si la descarga pudo hacerse correctamente (Código HTTP 200)

Ejemplo - Partial Render

```
function loadClick(event) {
    event.preventDefault();
    let container = document.querySelector("#use-ajax");
    fetch("https://web-unicen.herokuapp.com/api/html?")
        .then(response => {
            if (response.ok) {
                response.text().then(text => container.innerHTML = text);
            } else
                container.innerHTML = "<h1>Error - Failed URL!</h1>";
        })
        .catch(error => {
            console.log(error);
            container.innerHTML = "<h1>Error - Connection Failed!</h1>";
        });
}
...
```

Ejemplo 3 (error en nombre de archivo / apagar local server)
https://codepen.io/webUnicen/pen/Paopjo

fetch().then().then()

¿Qué está ocurriendo en cada llamado a la función **then()**?

```
fetch('/file.html')
  .then(r => {
       return r.text();
  })
  .then(html => {
      console.log(html);
  })
  .catch(e  => {
    console.log("Booo");
  })
```

Respuesta de la solicitud fetch

Procesamiento de la respuesta
(Nos da otra promesa)

Respuesta procesada

Error de conexión

AJAX es Asincronico

Repaso

SINCRÓNICO

Meto comida en el horno → Miro el horno hasta que esté lista → Ceno

Se hace la comida

ASINCRÓNICO

Meto comida en el horno y prendo timer → Preparo la mesa → Cargo la serie → ... → Ceno con una serie

Se hace la comida → RING! ← Responde la promesa

¿Cómo agrego el cartelito de “Loading...”?

Orden de ejecución

Pensemos en el orden de ejecución:

- Primero se hace la promesa
- Luego se sigue ejecutando
- Finalmente se ejecuta la resolución de la promesa

Ejemplo - Partial Render

JS

```
function loadClick(event)
{
  event.preventDefault();
  let container = document.querySelector("#use-ajax");
  container.innerHTML = "<h1>Loading...</h1>";
  fetch("https://web-unicen.herokuapp.com/api/html?").then(
        function(response){
              response.text().then(t  =>
                    container.innerHTML = t)
        });
  //podría ir aca el “Loading…” también?
}
let jsloads = document.querySelectorAll(".js-load");
jsloads.forEach(e=> e.addEventListener("click", loadClick));
```

Ejemplo 4 - https://codepen.io/webUnicen/pen/JZjWwx

¿El botón que cargamos en el fragmento de HTML anda?


encuentro
www.EncuentroEnElEstudio.com

Recuerden: esto es asincrónico

¿Que pasa si mediante AJAX cargo un botón que tiene una función JS asociada?

Si asigne el comportamiento antes usando “querySelector(...).addEventListener”, esta línea se ejecutó antes de que el botón exista.

El botón no tiene función asignada.

Como se soluciona?

- Asignar el handler cuando creo el botón en el html

Ejemplo donde el botón cargado recarga el DIV: http://codepen.io/webUnicen/pen/VjwgpO

Ejemplo - Partial Render

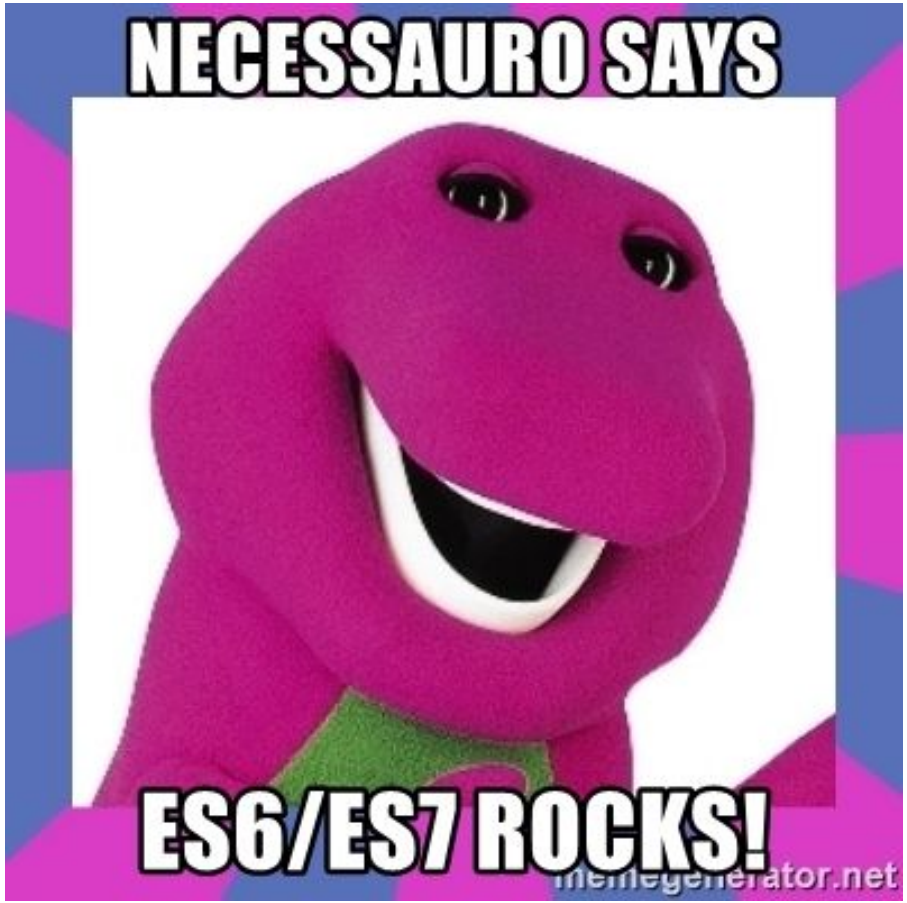
```
function processText(t) {
  let container = document.querySelector("#use-ajax");
  container.innerHTML = t;
  container.querySelectorAll(".js-load")
           .forEach(b=> b.addEventListener("click", loadClick));
}

function loadClick(event)
{
  event.preventDefault();
  document.querySelector("#use-ajax").innerHTML = "<h1>Loading...</h1>";
  fetch("https://web-unicen.herokuapp.com/api/html?").then( function(response){
      if (response.ok) {
        response.text().then(processText);
      }
      else
         ...
```

Ejemplo 5: - https://codepen.io/webUnicen/pen/PaomYG

AWAIT!!! ASYNC!!!

Promesas con async/await


NECESSAURO SAYS
ES6/ES7 ROCKS!

AWAIT!!! ASYNC!!!

- En **ES8** se incorpora dos palabras reservadas para facilitar la escritura de código con promesas

ASYNC

- Hace que una función devuelva una promesa
- El return se encapsulará en la promesa automáticamente

AWAIT

- Desencapsula el contenido de una promesa.
- Se reescribe como el THEN de la promesa
- Solo puede usarse dentro de funciones ASYNC

https://javascript.info/async-await

Sin await/async

```
function loadClick(event) {
 event.preventDefault();
 let container = document.querySelector("#use-ajax");
 container.innerHTML = "<h1>Loading...</h1>";
 fetch(url).then(
   function (response) {
     if (response.ok) {
       response.text().then(t => container.innerHTML = t);
     }
     else
       container.innerHTML = "<h1>Error - Failed URL!</h1>";
   })
   .catch(function (response) {
     container.innerHTML = "<h1>Connection error</h1>";
   });
}
```

Con await/async

```
async function load2(event) {
  event.preventDefault();
  let container = document.querySelector("#use-ajax");
  container.innerHTML = "<h1>Loading...</h1>";
 try {
    let response = await fetch(url);
    if (response.ok) {
      let t = await response.text()
      container.innerHTML = t;
    }
    else
      container.innerHTML = "<h1>Error - Failed URL!</h1>";
  }
 catch (error) {
    container.innerHTML = "<h1>Connection error</h1>";
  };
}
```

Veamos documentación

Veamos la documentación de MDN sobre el tema:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

An async function is a function declared with the `async` keyword. Async functions are instances of the `AsyncFunction` constructor, and the `await` keyword is permitted within them. The `async` and `await` keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

Async functions may also be defined as expressions.

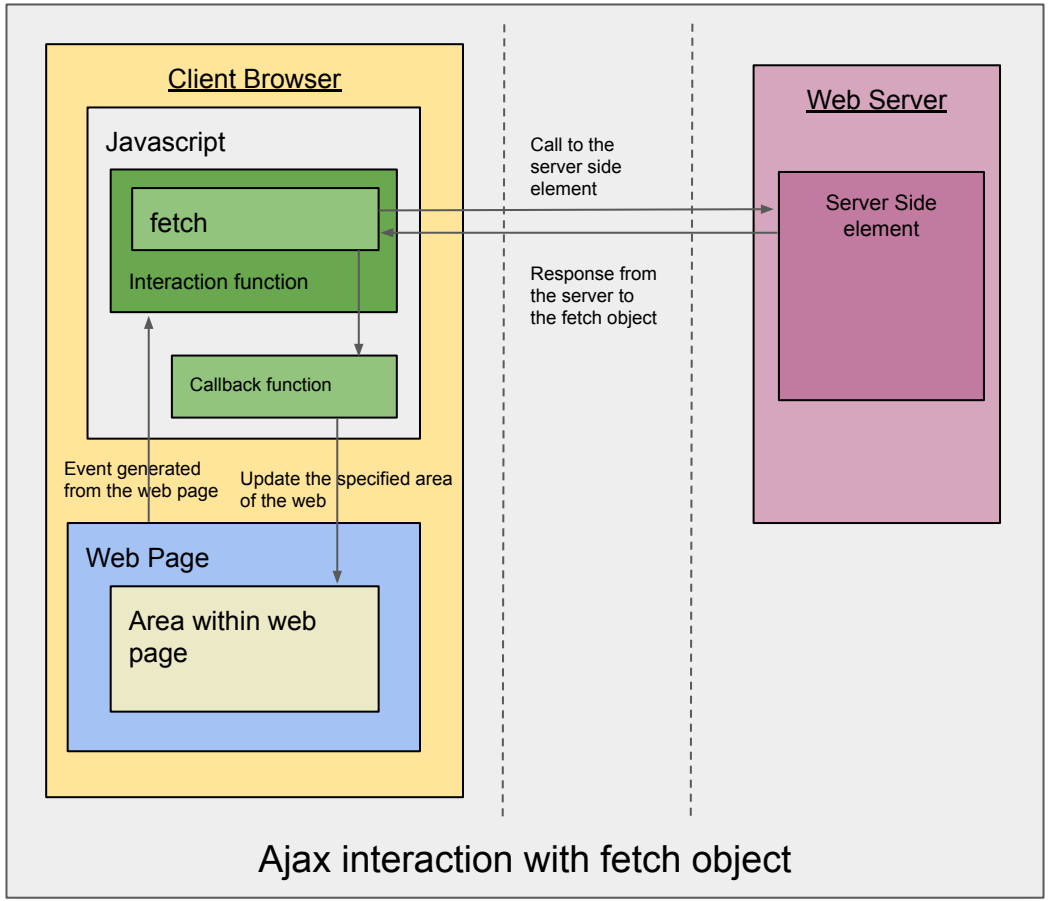
JavaScript Demo: Statement - Async

```
1 function resolveAfter2Seconds() {
2   return new Promise(resolve => {
3     setTimeout(() => {
4       resolve('resolved');
5     }, 2000);
6   });
```


vevo

Nav con partial render

Una llamada asincrónica

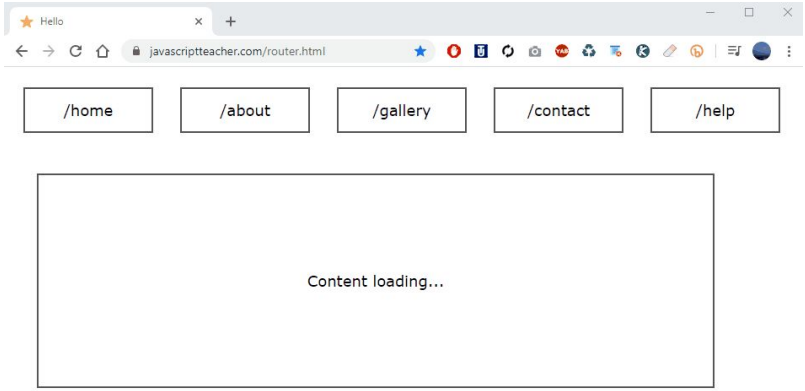


Qué podemos hacer con esto?

Podemos hacer navegación bajando las cosas a medida que la necesitamos sin que la página se refresque.

Al hacer click en cada boton del nav:

1. Colorea el nav (aplicando una clase)
2. Genera un Partial Render para descargar el contenido
    a. Luego lo muestra en un div preparado para eso



URL Push state

Si navego con partial render deja de andar la URL!

Debo programar cosas extras (JS y servidor) para que funcione completo.

Con Javascript puedo cambiar la URL sin cambiar de página

```
window.history.pushState(data, "Titulo", "nuevaUrl");
```

Permite asociar una URL a lo que estamos viendo (en este estado).

La URL no existe como archivo en el servidor. Si queremos que la URL funcione tenemos que implementarlo en el server (y según como lo haga algo en JS).

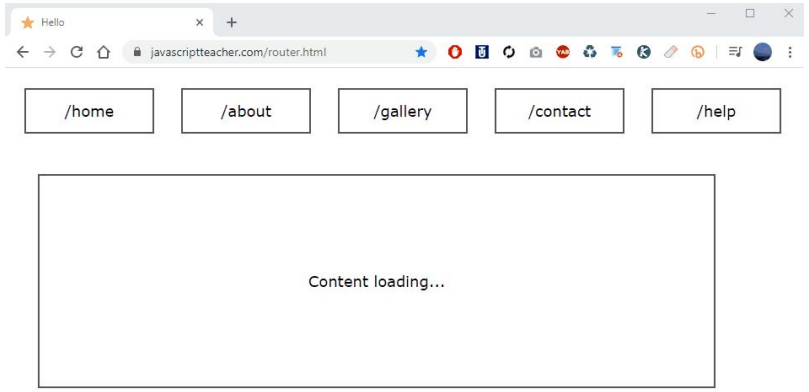
https://developer.mozilla.org/en-US/docs/Web/API/History_API/Example

Router JS

Al hacer click en cada boton:

1. Colorea el nav (aplicando una clase)
2. Genera un Partial Render para descargar el contenido
3. Con history.push agrega eso a la historia con el ID del botón apretado

Al hacer click en “página anterior” o “página siguiente” se va a llamar el evento history.pop, donde podemos usar el ID que guardamos



https://jstutorial.medium.com/making-your-own-vanilla-js-router-2e621ffdee88

Usando un servidor web para desarrollo

- Por políticas de seguridad, muchos navegadores no permiten hacer “fetch” de archivos locales (file://....), es decir archivos de tu disco rígido.

- Si lo permitieran cualquier página podría leer tus archivos.
- Por eso es obligatorio usar un servidor web para desarrollar esto.
- En esta cátedra usamos XAMPP (Apache + ...)
- LiveServer (VS Code)

Referencias

- http://api.jquery.com/jquery.ajax
- https://eamodeorubio.wordpress.com/category/webservices/rest/
- https://developer.mozilla.org/es/docs/AJAX
- http://www.restapitutorial.com/lessons/whatisrest.html
- “BulletProof AJAX” Jeremy Keith

AHORA LES TOCA PRACTICAR :D