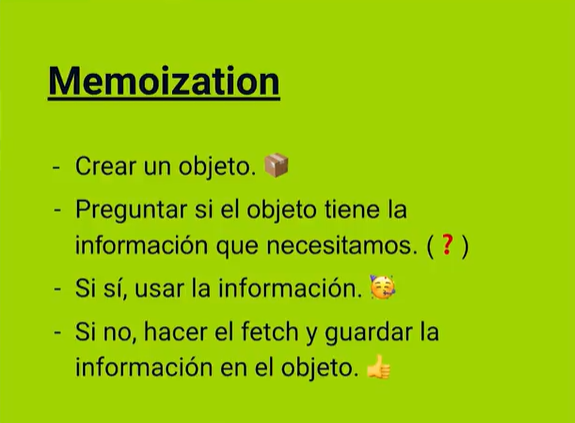
CURSO PROFESIONAL DE API REST CON JAVASCRIPT

**Caché vs. memoria**

MEMORIZATION

Que ya no sea el navegador que haga ese condicional con el cache, si no que seamos nosotros en nuestro código JS, con objetos, arrays, etc. Que guardemos esa información, en nuestro propio código, y que también podamos validar si volvemos a consumir la información o si utilizamos la información que ya tengamos ahí.

Guardar los cálculos que ya hicimos para no repetirlos.



La información se pierde cuando recargamos la página.

# Debuggeando caché y networking

1. Como frontend developers no podemos hacer que las APIs REST respondan más rápido, pero sí podemos:

Evitar que los usuarios carguen la misma información varias veces.

1. Los navegadores por defecto pueden aplicar caché / disk memory para guardar los resultados de nuestras consultas HTTP.

Verdadero.

1. ¿Cuál es un MAL caso de uso de caché o memoization?

App de chat realtime.

1. ¿Cuál es un buen caso de uso de la memoization?

Al hacer un cálculo Fibonacci

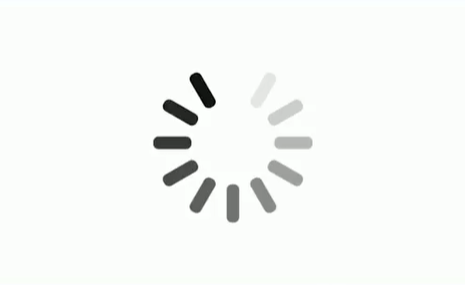
# Loading spinners vs. loading skeletons

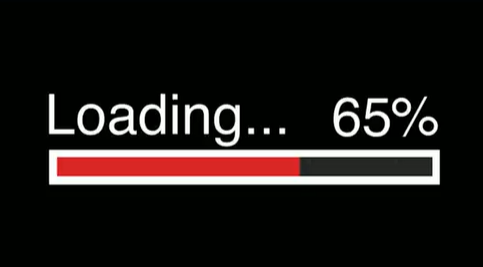
Una pantalla esqueleto (skeleton screen) es esencialmente una estructura de la aplicación. Son un reemplazo para el spinner de carga tradicional.

Estos ayudan a mejorar la experiencia de usuario, haciendola ver con un mejor performance.



Loading Spinners



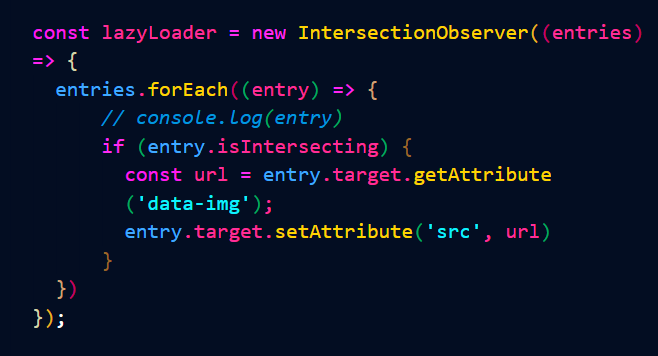


# Intersection Observer

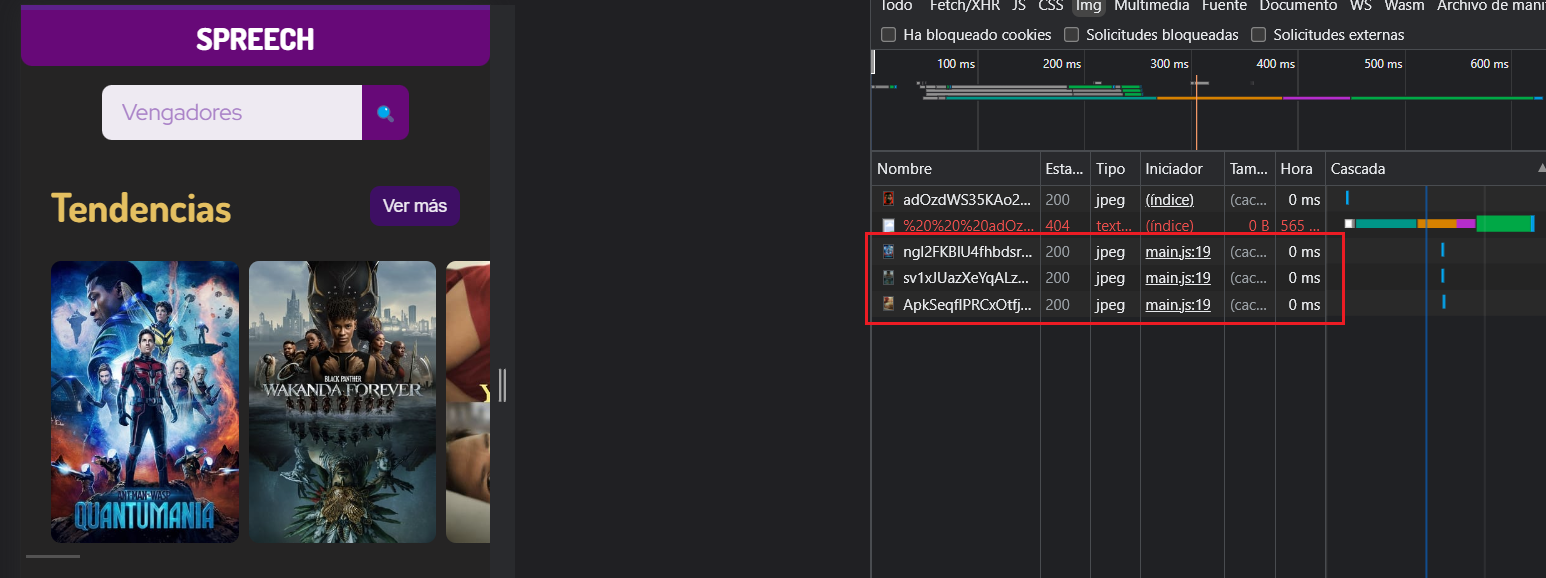
Esta función en este caso, nos va a permitir ver en el viewport de los usuarios, quiere decir la parte de la pagina que estan viendo los usuarios, y si hay imágenes en esta, cargarlas únicamente porque los usuarios estan en ese, para asi, si llegan a seguir bajando o van a otro lugar carguen las nuevas imágenes.

NO PARA QUE CARGUEN TODAS DE UNA!.

# Lazy Loading



1. Crear la instancia del intersection observer con > new
   1. Esta lleva un callback, se le dio un argumento que es > entries
   2. Luego se llama al argumento + el método forEach y se le pasa otro callback, este lleva el argumento entry (que es cada uno de los método que tiene Intersection Observer)
2. Se pregunta si, es interceptado (entry + el atributo isIntersecting)
   1. Se crea la variable > url, que busca el atributo o clase de un div, que es => data-img, y lo cambia por => src.



Esto hace que, como vemos estamos en la pagina principa, y es lo único que se nos muestra, en el apartado de red, solo se carga 3 imágenes, pero si damos scroll, se van cargando todas a medida.



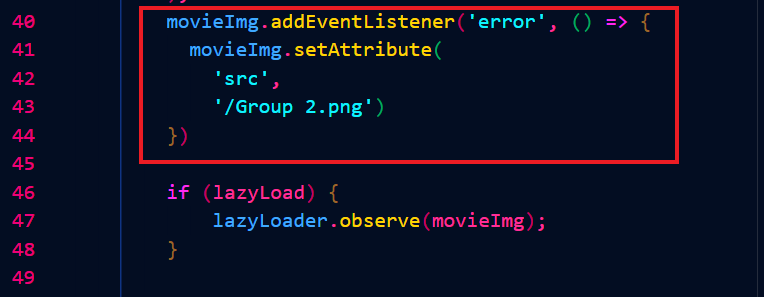
En la función > createMovies, se le pasa un tercer argumento, que es si, se quiere que sea interceptado dicha vista o no.

En el segundo recuadro, preguntamos si lazyload, no es => data-img, entonces que sea => src.

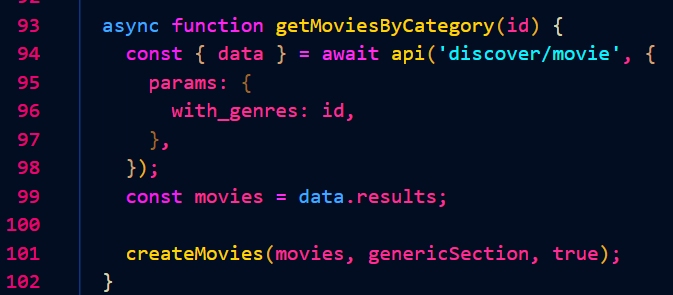
En el tercer cuadrado, se valida si lazyLoad quiere ser usado, si si, se llama a la instancia + el método > observer, y se le pasa el contenedor de la imagen.

De esta forma observa las imágenes.

# Imágenes por defecto



En la función > createMovies, se le agrego a la variable o contenedor de la imagen, que cuanto tuviera el evento de error, cambiaramos el source o la fuente de la imagen, cuando la pelicula no salga, para que muestre la imagen de error y no solo el alt.



Y se le dijo a la función createMovies que desactivara el intersection Observer, en la función > getMoviesByCategory

QUIZ

1. ¿Qué es Lazy Lading?

Indicarle a nuestras imágenes que no carguen todas desde el principio, sino únicamente cuando los usuarios lleguen a esa parte de la app.

1. Tu aplicación es una reserva de tiquetes para un circo. ¿Es buena idea contar chistes en la pantalla de carga?

Sí, va con el tono de la aplicación.

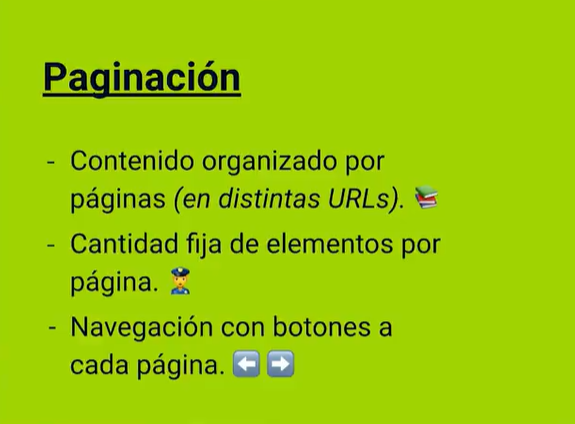
1. ¿Qué es el churn rate?

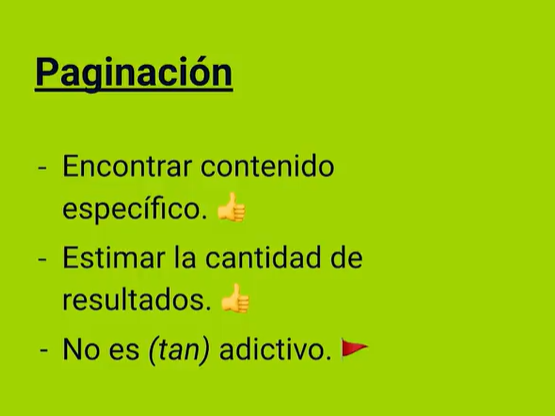
La cantidad de usuarios que entran a nuestra app pero luego la abandonan.

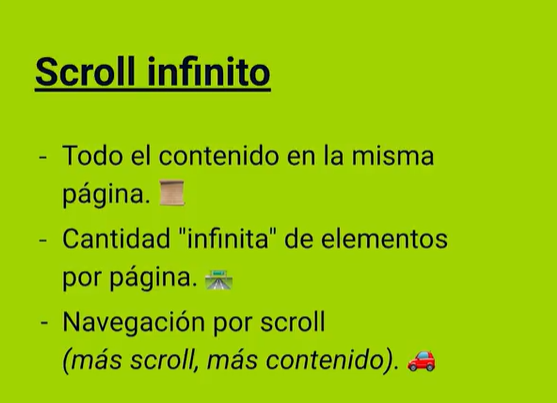
1. ¿Por qué son importantes las pantalllas de carga?

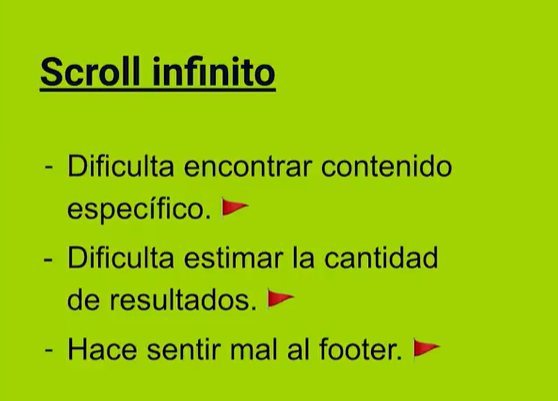
Para que los usuarios sepan dónde está cargando el contenido que buscan.

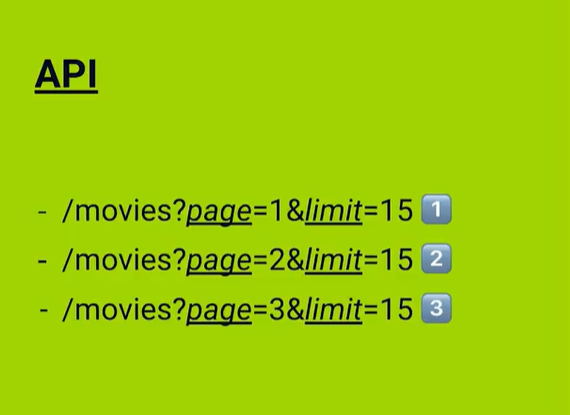
# Scroll infinito vs. paginación



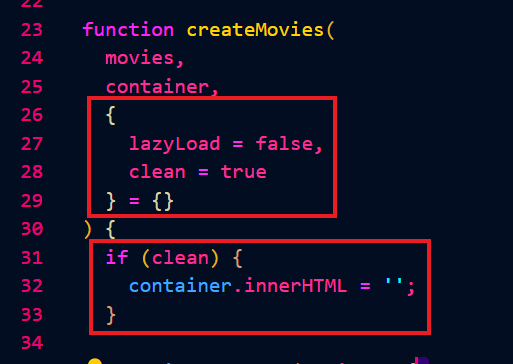








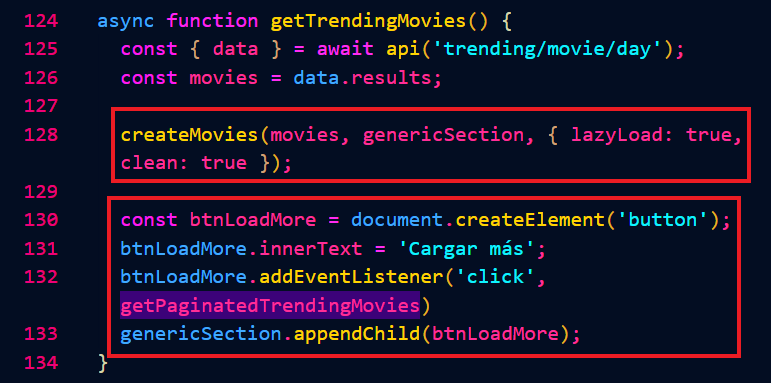
# Botón de cargar más



En la función > createMovies, le colocamos un nuevo argumento, en este caso es clean (Es para limpiar el contenedor dependiendo el caso)

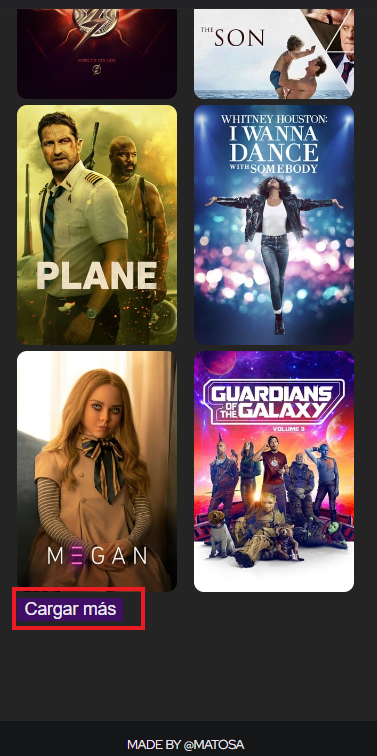
En la condicional, verificamos si clean es verdadero, entonces se limpia el contenedor.

* Los argumentos > lazyLoad y clean, se colocan dentro de un objeto, para que al llamarlos queden como un atributo.

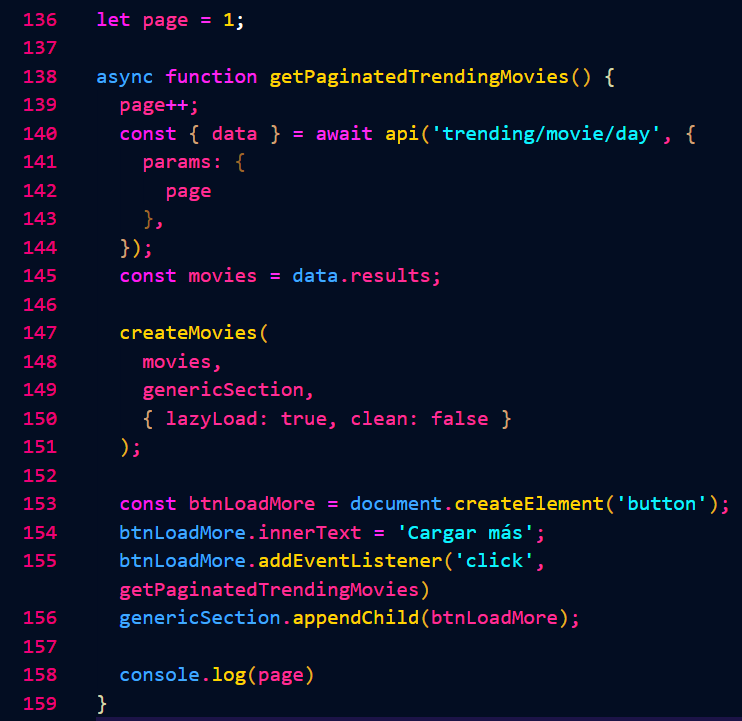


En esta función, creamos el botón con > createElement

* Le agregamos texto
* Le agregamos una función
* Y lo agregamos a la sección



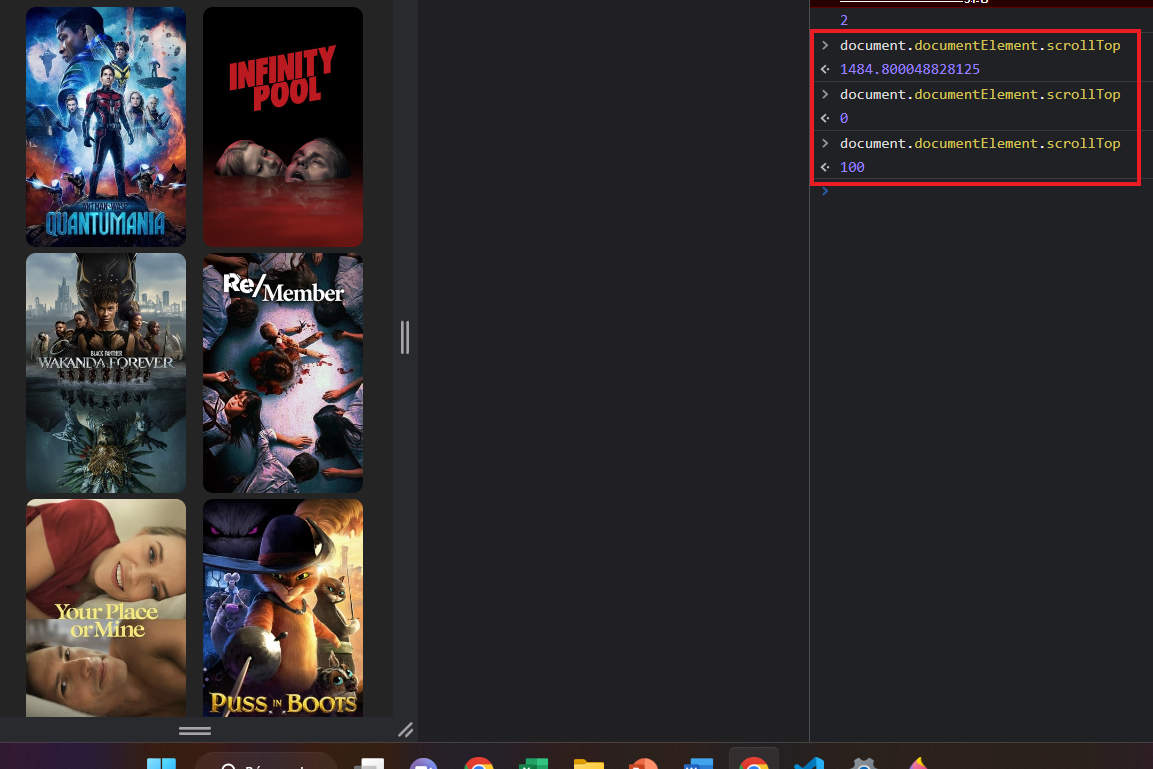
La función del botón



1. Cogemos la misma llamada a la api de getTredingMovies, pero esta vez con los params, y solo agregándole page (paginación)
2. Llamamos a la función > createMovies, aquí llamamos a los argumentos lazyLoad y clean, se llama a clean para que no limpie el contenedor
3. Se agrega el mismo botón anterior para seguir cargando mas películas
4. La variable > page, se inicializa por fuera en uno, quiere decir apenas carga la pagina, es la pagina 1, si, se le da al botón de cargar mas, se inicializara en 2 y sucesivamente.

# Infinite Scrolling: evento de scroll

3 propiedades con las que vamos a poder calcular, cuando los usuarios lleguen al final del scroll de la ruta de la navegación en la que estemos.



(document + documentElement + scrollTop)

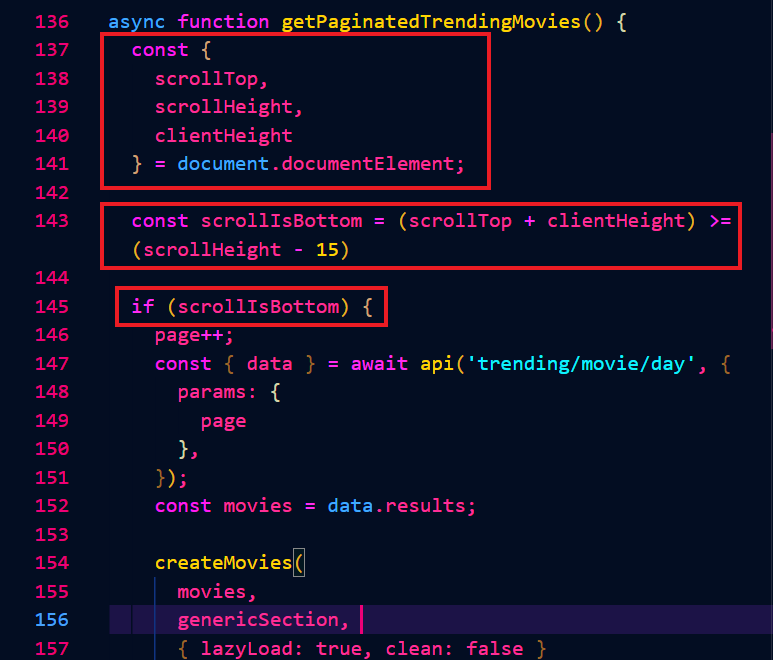
Es para saber en que pixeles esta la vista del usuario.

(document + documentElement + clientHeigth)

Es para saber el alto del dispositivo de cada usuario.

(document + documentElement + scrollHeigth)

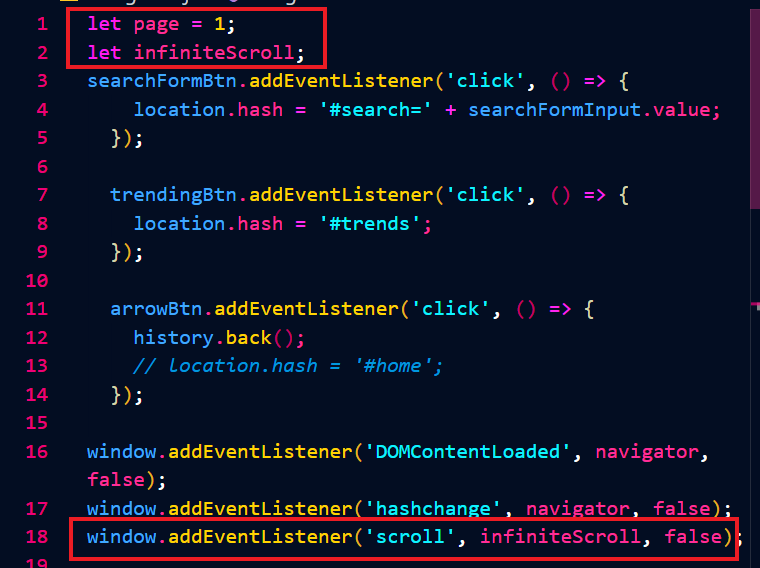
Nos dice, cuanto scroll podemos hacer en la aplicación.



Para saber cuando el usuario hizo scroll y llego abajo y luego mostrar mas películas, se hace una operación matemática primeramente de las 3 propiedades

1. Se suma el scrollTop + clienteHeight (La suma del scroll que hace el usuario + el alto de la pantalla del usuario) si esto es mayor o igual a…
2. Al scroll que podemos hacer en la pagina, restándole – 15 (que son 15 pixeles)

* Verificamos si lo anterior es verdadero, entonces que me siga mostrando imágenes, que es lo que se hace en la función getPaginatedMovies.



En el archivo de navigation.js

Colocamos las variables page e infiniteScroll

Y agregamos un evento cuando se le de scroll a la aplicación, a este le agregamos la variable infiniteScroll (esta va cambiando cuando hacemos scroll)



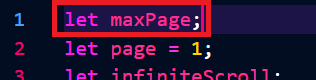
En la función navigator

Verificamos comenzando si la variable infiniteScrol esta, le eliminamos el evento, y aparte le damos undefined a la variable.

Luego al final con el condicional, lo volvemos a colocar.

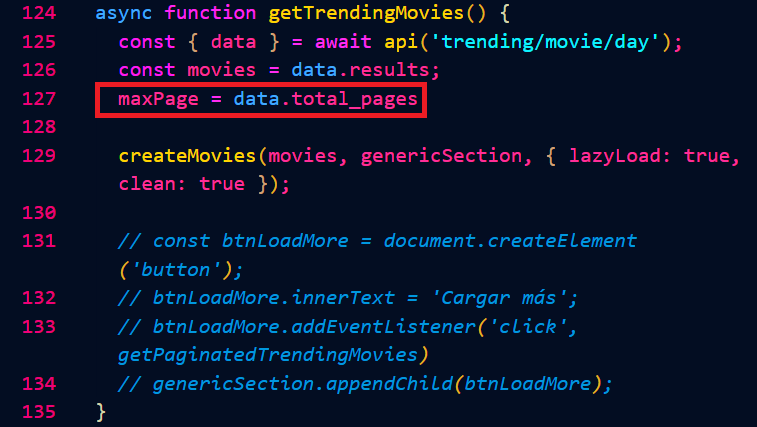
Passive lo que hace es evitar el llamado de preventDefault() en el caso de que este existiese en la función llamada por el Listener. En los navegadores que usa la gente normal el valor por defecto es false por lo que no se aplica, pero en el caso de Safari e Internet Explorer el valor por defecto es true. Por lo que supongo que es recomendable ponerle un valor para que el código se ejecute igual en todos los navegadores.

# Infinite Scrolling: limitando la carga de datos



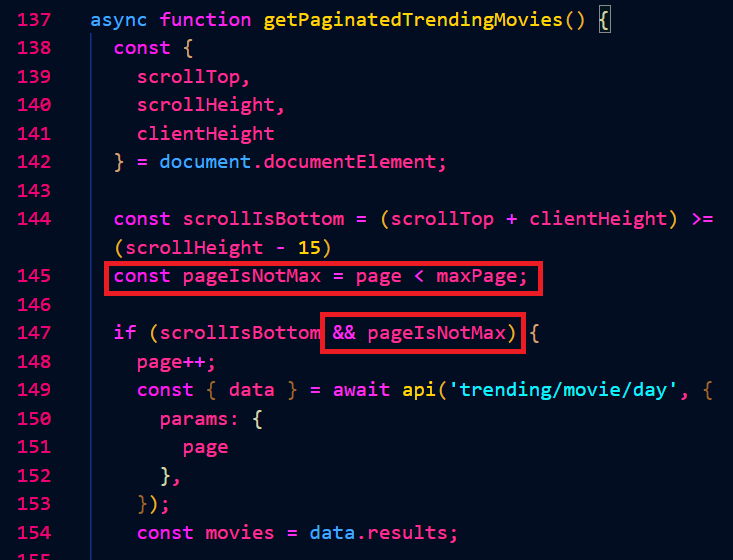
Se creo la variable en el archivo de navigation.js

(Esta variable representa la máxima paginación que son 1000).



En esta función, se llama a esa variable, y se le da por valor, que sean las paginas totales que nos permite la api.

Esto se hace llamando a data (Información de la api) + el atributo que nos dan en la api => total\_pages (1000)



En esta función

Creamos en la variable pageIsNotMax, que la variable page es menor a el máximo de paginas.

Y se coloca en validación si esto es asi, entonces, no podrá imprimir mas de la cuenta.

# Infinite Scrolling: closures de navegación

1. ¿Qué característica(s) debe soportar nuestro backend / API REST para que podamos implementar scroll infinito?

page & limit

1. ¿Qué es más adictivo?

Hacer scroll

1. ¿Qué estrategia usarías si en tu aplicación es prioritario encontrar contenido específico fácilmente?

Paginación.

# Local Storage vs. API real

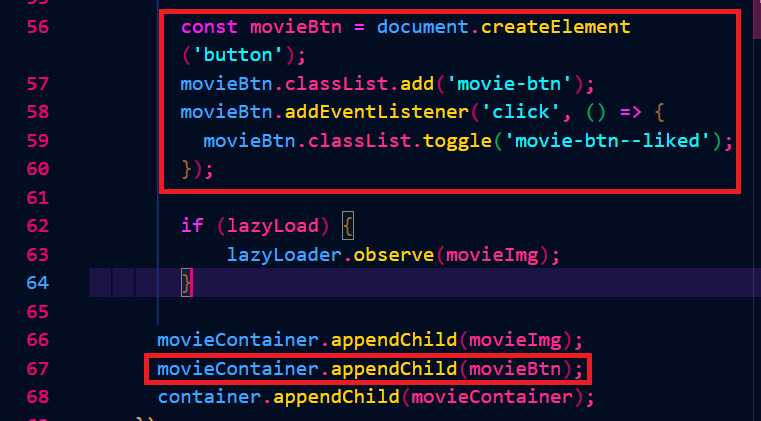
Datos interesantes

* El localStorage guarda de 5 a 10mb de datos (dependiendo del navegador)
* No necesita bases de datos ni backend
* Excelente para probar ideas y hacer testing de funcionalidades
* Solamente guarda STRINGS
* Se guarda por cada navegador
* Es inseguro para guardar información
* No requiere autenticación
* No permite compartir o recuperar sesión.





# Botón de like



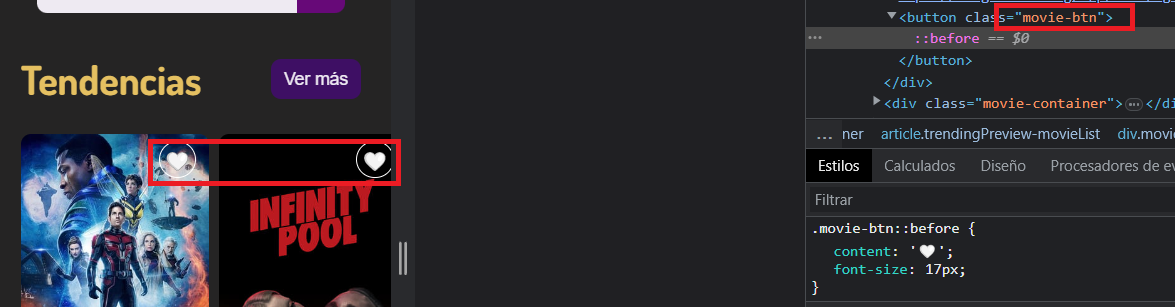
En la función > createMovies creamos el botón con > createElement

* Se le dio una clase
* Se le agrego un evento de click, el cual se le dijo que cuando le den click, cambie el nombre de la clase

Esto se hace con > toggle.

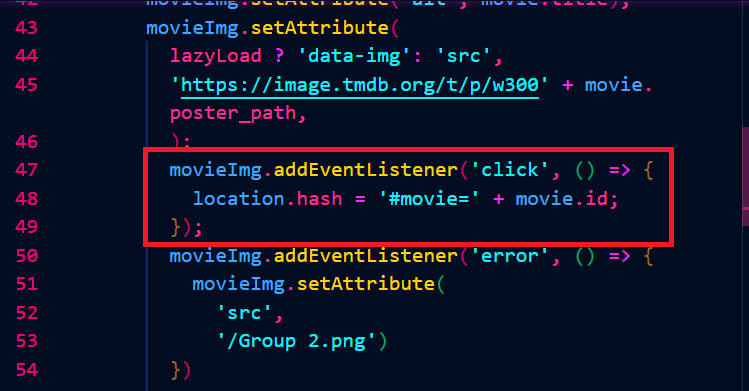
* Luego se añade al contenedor.

Nombre de la clase sin haberle dado click



Al darle click en el botón, cambia el nombre de la clase y el botón





Siguiendo en la misma función, se coloca ese evento, pero al contenedor de la imagen, para que no haya problema cuando se le de click al botón.

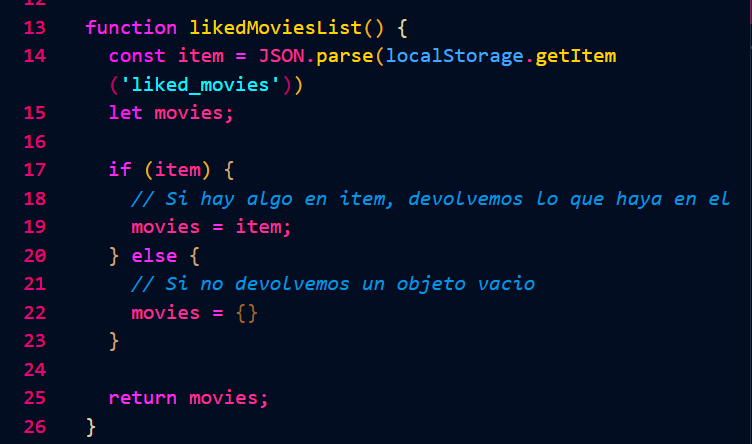
# Guardando películas en Local Storage

localStorage.getItem (Para leer información)

localStorage.setItem (Para guardar información)

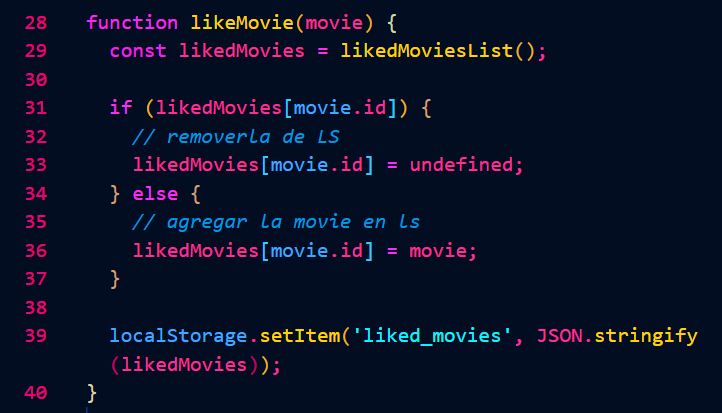
localStorage.setItem (Para eliminar información)

Es solo para guardar strings.



Se creo esta función (likedMovieList)

* La variable > item, significa que va a parsear (convertir de string a objeto) ese atributo el cual es > liked\_movies
* Se crea la variable > movies, que va a ser el nuevo objeto o contenedor de la información de lo que trae la variable => item.
* Se valida si hay algo en item, devolvemos lo que haya, si no devolvemos un objeto vacio.



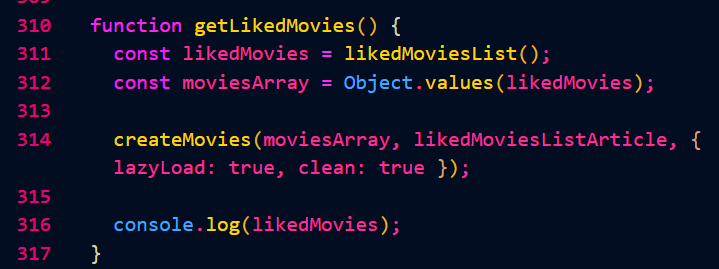
En esta función, la cual tiene 1 parámetro

1. Se llama a la anterior función, y se guarda en la variable > likedMovies
2. Se pregunta si la variable anterior, en las películas, en el atributo > id, lo primero que va hacer es eliminarlas del objeto y si no, agregarla al nuevo objeto.
3. Se cambia en el localStorage con > setItem, el atributo > liked\_movies, y se stringyfy (pasar de objeto a string), la variable likedMovies.

# Lista de películas favoritas

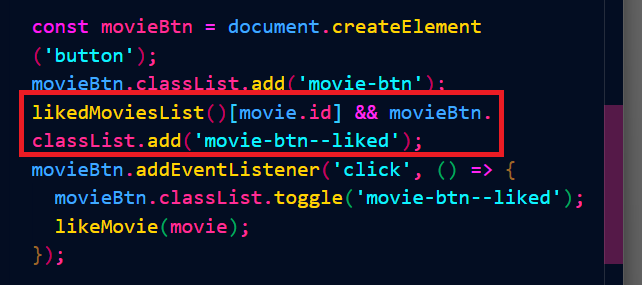
Object.values()

Este método nos ayuda a crear un array con todos los valores de un objeto.



Esta función es para obtener las películas que se les haya dado like, y imprimirlas en pantalla.

1. En la variable likedMovies se llama a la función => likedMovies
2. En la variable moviesArray, se llama a > Object + values y se le pasa la variable anterior (Esto es para cree un nuevo array de las películas que les han dado like)
3. Luego se llama a la función de crear películas, y se le pasa la variable moviesArray + likedMoviesListArticle (Que es el contenedor)



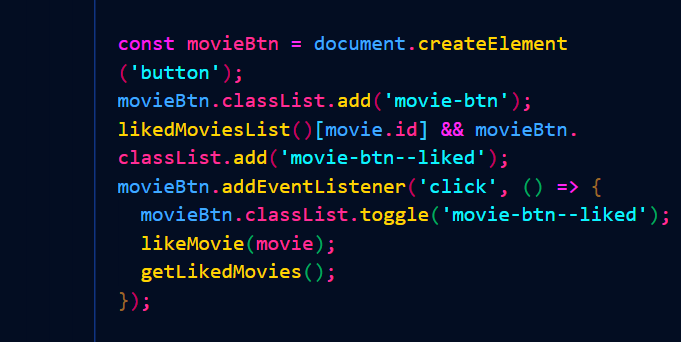
En la función de createMovies

1. Se llama a la función > likedMoviesList + dentro de un array, movie + id (ósea el id de cada pelicula) y (&&) la clase del botón

(Esto lo que hace es colocar el estado que tiene el botón cuando tiene esa clase)

En la función homePage en navigation, se llama a la función likedMoviesList.

Para cuando al darle me gusta a una película, inmediatamente se vea reflejado en la sección de favoritos



Llamamos a la función > getLikedMovies (Que es la que crea las películas favoritas)

QUIZ

1. Guardar mi API KEY en Local Storage es una buena práctica.

NOOOOOOO

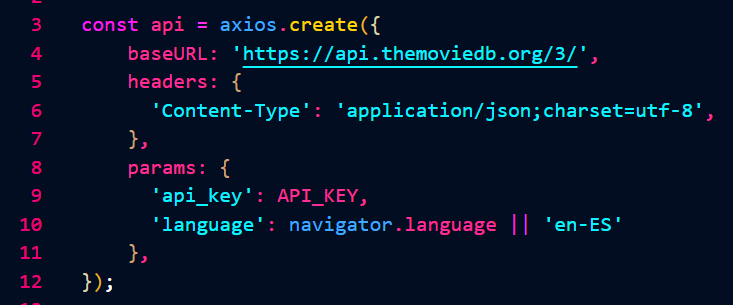
1. Local Storage es mas rápido que una API REST.

Verdadero

1. Tu aplicación es un chat realtime. ¿Qué característica puedes desarrollar con Local Storage?

Elegir el idioma de la aplicación.

# selección de idioma



Para agregar el lenguaje dependiendo el navegador del usuario

1. En los params de la axios, se pasa asi mismo > language como una key + navigator (que es el navegador y lee sus propiedades) + language (que es el que lee el lenguaje que tiene el usuario) => || (y si no) que sea en ingles (en) o español (ES).