**PROGRESSIVE WEB APPS**

Son aplicaciones que simulan ser nativas, como aplicaciones de escritorio o mobiles, pero desde la web.

ENFOQUE 🡪 MOBILE FIRST

Para que se considere una PWA o el navegador la interprete como tal debe:

* Correr bajo un protocolo HTTPS o estar en un localhost
* Contar con un service worker
* Tener un archivo manifest.json

Las principales ventajas, son:

* Instalables
* Pesan muy poco, hasta comparado con apps nativas
* Seguras
* Responsivas
* Acceden a recursos nativos del dispositivo, como cámara, alertas, notificaciones, etc.
* Corren sin conexión. SI el usuario realiza cambios que deban reflejarse en una DB mientras no tiene conexión, se guardan en una especie de DB local para subirse cuando se conecte nuevamente a la red.
* Se actualizan constantemente, ya que “se manejan por cache”

**MODELO CLIENTE SERVIDOR**

La principal diferencia entre HTTP y HTTPS (el que usan las PWA) es la seguridad. En HTPPS se manejan los datos de forma cifrada, por tanto en el traspaso de los mismos, si se llegase a interceptar, no habría pérdidas ya que los datos no serían manipulables, entendibles o explotables.

La razón por la que es necesaria usar HTPPS en PWA es que estas utilizan Service Worker y este se considera “muy poderoso”, por tanto, si se deja este en riesgo, puede modificarse completamente la aplicación, logrando resultados no muy deseados.

**SERVICE WORKER**

Es una característica de los navegadores modernos, no todos estos la soportan.   
ES una especie de intermediario entre el servidor(la app) y el cliente (el hardware). Analiza las peticiones y las características físicas del cliente para operar en base a estas.

El service worker consta de un archivo JS, completamente independiente a la app que se encarga de guardar datos en cache, actualizarlos, lanzar notificaciones push y guardar acciones cuando estén sin conexión, para luego “lanzar” estas acciones de forma definitiva.

Es necesario que este fichero corra bajo protocolo seguro porque este es el que se descarga la primera vez que el usuario descarga la app, por tanto, si se corroe éste, hasta que no se borre manualmente el caché no se podrá obtener el archivo “no corrupto”.

Cuando se inspecciona un sitio web, como el ejemplo realizado en react del contador, en la pestaña **application** pueden encontrarse los apartados de: Manifest, Service Worker y Storage, Además de Cache Storage.

En la pestaña de network, pueden visualizarse las peticiones realizadas, además de su estado, su tamaño y el tiempo que tardaron. Si estas se almacenaran en el cache, la velocidad de respuesta de la petición de la app sería mayor.

**SERVICE WORKER**

Se puede decir que estos reaccionan en base a eventos.

Existe un objeto en todo sitio web, proveído por el navegador, al que podemos acceder desde la consola llamado **navigator** este, contiene **serviceWorker**. Para comprobar desde nuestra app que podamos utilizar service worker, esto es lo que se debe hacer.

En un nuevo archivo, que podría llamarse register.js y encontrarse al mismo nivel que el index, agregar:

if("serviceWorker" in navigator){

    navigator.serviceWorker.register("./sw.js");

}

En este caso, se “delega” el registro a otro archivo, sw.js, el que también debe ser creado en el directorio que se especifique.

Una vez hecho esto, podemos dirigirnos en nuestro sitio a la pestaña Application y al apartado de Service Worker para ver reflejados los cambios.

**INSTALANDO SW**

Para utilizar correctamente service worker, la idea es que se “cacheen” los recursos necesarios y luego, cada vez que se carga se compruebe por las actualizaciones de estos recursos.

Lo primero es lograr “cachear” los elementos. Esto se realiza desde el archivo **sw.js**, al cual se le delegó la acción desde **register.js**

Para simplificar el código, los recursos a cachear y el “name” del cache se guardan en constantes, de esta forma.

const CACHE\_ELEMENTS = [

    "./",

    "https://unpkg.com/react@17/umd/react.development.js",

    "https://unpkg.com/react-dom@17/umd/react-dom.development.js",

    "https://unpkg.com/@babel/standalone/babel.min.js",

    "css/main.css",

    "components/Contador.js",

    "index.js"

];

const CACHE\_NAME = "v1\_Cache\_contador\_react";

además, cuando en estos casos, por convención se usa **self** en lugar de **this**. Para esto es necesario hacer:

const self = this;

Luego, se agrega un escuchador al “primero” de los “eventos fundamentales”.

self.addEventListener("install", (event) => {

    event.waitUntil(

        caches.open(CACHE\_NAME)

        .then((cache) => {

            cache.addAll(CACHE\_ELEMENTS)

            .then(()=> {

                self.skipWaiting()

            })

            .catch((error) => console.log(error));

        })

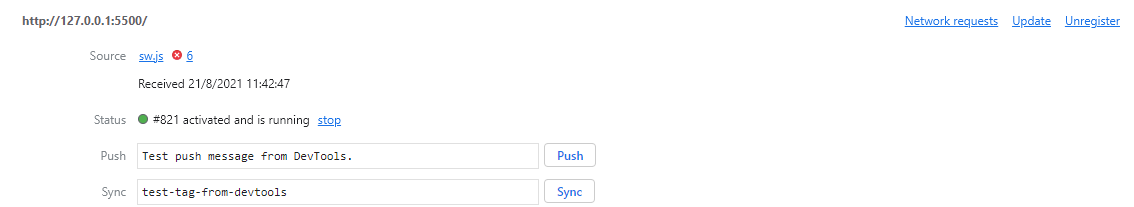
    );

});

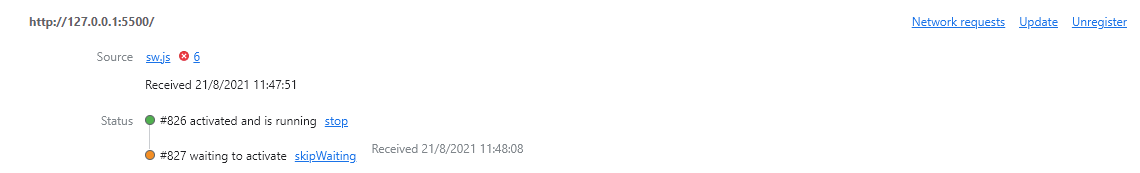
EN este código:

* Se escucha por el evento install y se declara una arrow function anónima
* Se invoca a **waitUntil()** del evento el cual espera hasta que se ejecute **caches**.open. Caches es el objeto y open el método que “inicia” un nuevo cache con el nombre obtenido por param. EN este caso nuestra const CACHE\_NAME.
* Este evento open retorna una promise, por eso se continua con **.then()**. Esta recibe por parámetro (cache), que en realidad es el objeto cache creado con .open()
* A este, se le aplica **.addAll()** el cual agrega a este cache los elementos pasados en param, en este caso nuestra const CACHE\_ELEMENTS, que son las rutas de los recursos a cachear.
* Este evento también retorna una promise la cual ejecutará **self.skipWaiting()** para que cada que se introduzca se utilice el sw.
* Por ultimo se agrega un .catch, en caso de que alguna ejecución falle.

Para entender mejor el skipWaiting() podemos dirigirnos a la pestaña Application en inspeccionar y al apartado Service Workers



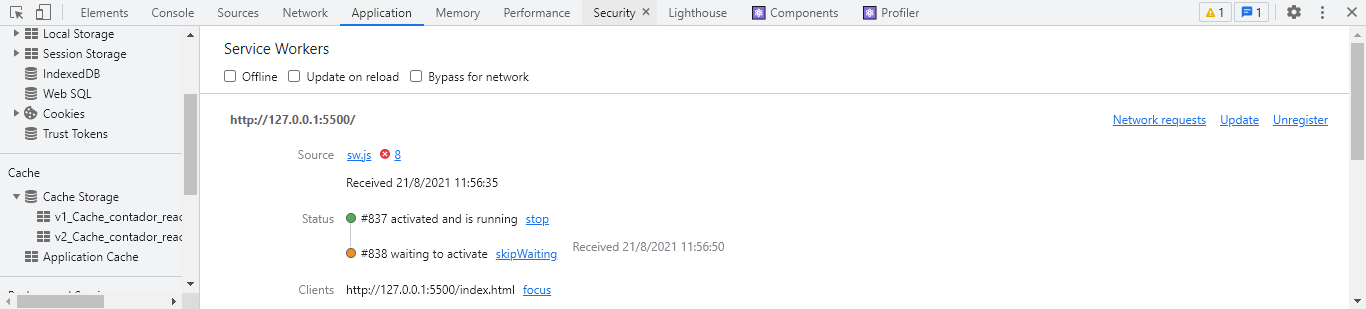
En status debería aparecer un service worker con la opción disponible **skipWaiting**



Algo así

**AGREGANDO NUEVOS CACHES Y ELIMINANDO VIEJOS**

Si por ejemplo, cambio el nombre en CACHE\_NAME, se genera un nuevo cache, almacenando a la par el anterior:





Lo que necesitaríamos en este caso, sería comparar el valor del último con los que hay almacenados para dejar solo este.

Para esto, es necesario realizar acciones en el evento **activate**

**ACTIVATE**

self.addEventListener("activate", (event) => {

    const cacheWhiteList = [CACHE\_NAME];

    event.waitUntil(

        caches.keys().then((cachesName) => {

            return Promise.all(

                cachesName.map(cacheName => {

              return (cacheWhiteList.indexOf(cacheName) === -1 && caches.delete(cacheName)

                    );

                })

            );

        }).then(() => self.clients.claim())

    )

});

en el evento actívate se realiza una funcion anónima y se obtiene por param datos del evento.



Para empezar se declara una const **cacheWhiteList** que consta de un [] de CACHE\_NAME.

Luego, al igual que en el evento install, se realiza el **waitUntil.** Dentro de deste, se invoca al objeto caches y a la funcion keys() que “llama” a todos los caches generados.

En la promise de este método, se obtiene **cachesName** lo cual me da un array con todos los caches en el service worker. De esto se retorna una nueva promise, la cual utiliza .all para “recorrer” todos los elementos.

Con .map recorremos cada uno de estos y retornamos en caso de que el índice de nuestro cache (**cacheWhiteList**) sea (-1, o sea no exista) y procede a eliminarle.

Por ultimo se “reclama” el cache desde la promesa de**cache.keys()**

**FETCH**

El siguiente evento a analizar es el **fetch**

*Contiene información sobre la búsqueda (fetch), incluyendo la petición (request) y cómo el receptor tratará la respuesta (response).*

Retorna o cachea lo que haya en la pestaña application, en el apartado Cache Storage en el cache determinado.  
EN caso de que exista alguna otra petición dinámica, la cachea junto con las demás.

Este es el código para el evento fetch:

self.addEventListener("fetch", (e) => {

    e.respondWith(() => {

        caches.match(e.request).then((res) => res ? res : fetch(e.request));

    });

});

Se agrega la funcion **respondWith** con un arrow function en la cual se invoca al objeto caches y se busca por coincidencias.

En e.request se toman cada una de las peticiones y, en caso de que esa respuesta sea valida, o sea, coincida con las que hay la retorne, y si no, que retorne fetch(e.request);

En caso de que en algun momento un código no funcione, puede dirigirse a la pestaña Applicaction en el apartado Storage y pulsar en **clear site data**

**METADATOS E ICONOS**

Lo siguiente a hacer es agregar metadata e iconos. Si se sigue el proyecto hasta el momento, debería generar error, ya que en el listener del fetch, aun faltan cosas por hacer. Si se quiere obviar el service worker para trabajar y no recibir errores en la pagina, lo que puede hacerse es comentar la línea en **register.js**

Las etiquetas meta que se modifican son para datos de la app en general, y para su ejecución en dispositivos móviles

    <meta name="theme-color" content="#333333" />

    <meta name="mobileOptimized" content="width" />

    <meta name="HandheldFriendly" content="true" />

    <meta name="apple-mobile-web-app-capable" content="yes" />

    <meta

      name="apple-mobile-web-app-status-bar-style"

      content="black-translucent"

    />

Para agregar los iconos:

    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />

    <link rel="apple-touch-icon" href="favicon.ico" type="image/x-icon" />

    <link

      rel="apple-touch-startup-image"

      href="favicon.ico"

      type="image/x-icon"

    />

Las ultimas 2 etiquetas corresponden a los iconos en dispositivos apple

Un buen lugar del cual conseguir iconos es **iconfinder**

<https://www.iconfinder.com/>

además cuenta en su búsqueda con iconos para uso comercial sin atribución.

**MANIFEST.JSON**

Este archivo es fundamental en las PWA ya que contiene muchas configuraciones necesarias con las que se inicia la PWA. Es un archivo.json que contiene como llave el nombre de las propiedades y como valor el valor para las mismas.

La parte mas “tediosa” de hacer un manifest.json es la de configurar los “icons”. Consta de un array dentro de la llave **icons**

Una forma de solucionarlo es usando

<https://www.pwabuilder.com/imageGenerator>

desde aquí se importa el archivo y se genera con las características necesarias.

El resultado es un .rar que contiene tanto los archivos de imagen necesarios como el código para insertar en **manifest.json**

Otro paquete es pwa-asset-generator

Es importante introducir bien los comandos para usar este package, se recomienda leer la documentación o ver el video del curso en cuestión

<https://www.npmjs.com/package/pwa-asset-generator>

**ARREGLANDO EL SW**

Dentro del listener del evento fecth, en el archivo sw.js hay un error que se soluciona modificando el código de la siguiente manera

self.addEventListener("fetch", (e) => {

    e.respondWith(

        caches.match(e.request).then((res) => {

            if(res){

                return res;

            }

            return fetch(e.request);

        })

    );

});

Anteriormente se usaba el respondWith con su contenido como arrow function, ahora se usa como un método

Dentro de este respondWith, se invoca el método **caches** y se busca que coincida con un request de los caches seleccionados.

Esto retorna una promise, si coincide con los caches seleccionados, se retorna ese cache, si no, se hace la petición del mismo.

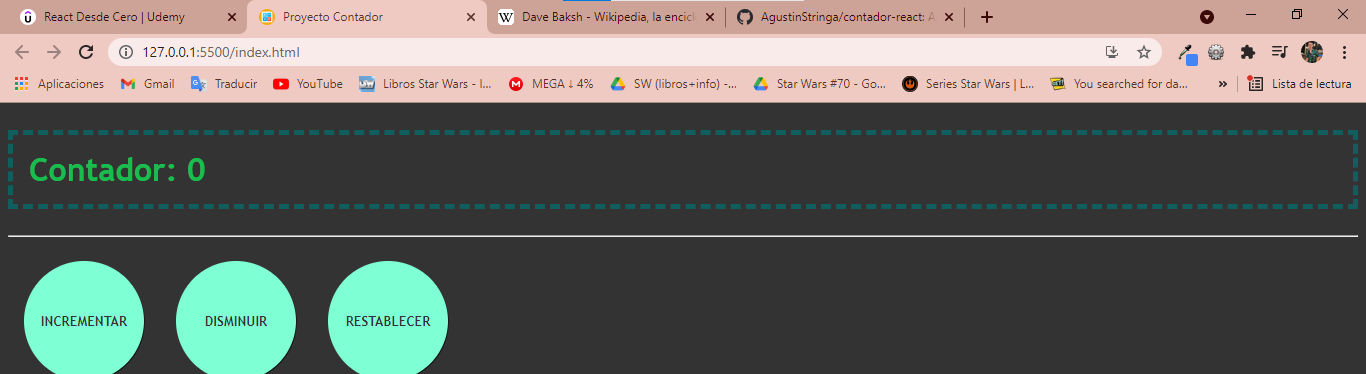
En resumen 🡪 compara los cache seleccionados 🡪 los actualiza.

**DESPLEGANDO EN GITHUB PAGES**

Para poder subir rápidamente la aplicación, es necesario crear un nuevo repositorio y agregar los archivos en la carpeta.

Si se despliegan los archivos directamente es mejor, ya que si se accede al url del repositorio, lo primero que detectará será el index.html

Si se abre desde el navegador se podrá encontrar enlace para instalar la aplicación



Instalar PWA



SI se abre desde un dispositivo móvil, aparecerá disponible la opción de agregar a la pantalla principal.

Además, cuando se suba a github pages y se abrá desde cualquier lugar, podrá encontrarse un candado al lado de la url que nos indica que se corre bajo el protocolo HTTPS.