SERVICE WORKER

Предназначен для кэширования запросов на сервер. То есть любой запрос на сервер кэшируется, а при отсутствии интернета приложение не ложится, а данные берутся из кэша.

SW запускается в worker контексте, поэтому он не имеет доступа к DOM и работает в потоке, отдельном от основного потока JavaScript, управляющего вашим приложением, а следовательно — не блокирует его.

Для того чтобы приложение стало более отзывчивым, нам нужно кэшировать все статичные файлы (JS CSS медиа-контент)

Подключение:

- в index.html добавить index.js:

**if** ('serviceWorker' **in** navigator) {  
 // Весь код регистрации у нас асинхронный.  
 navigator.serviceWorker.register('./sw.js')  
 .then(() => navigator.serviceWorker.ready.then((worker) => {  
 worker.sync.register('syncdata');  
 }))  
 .catch((err) => console.log(err));  
}

Создать фаил sw.js

self.addEventListener('install', (event) => {  
 console.log('Установлен');  
});  
  
self.addEventListener('activate', (event) => {  
 console.log('Активирован');  
});  
  
self.addEventListener('fetch', (event) => {  
 console.log('Происходит запрос на сервер');  
});

Есть разные варианты использования кэша. Самый популярный – сети и кэш – то есть если есть сеть то всегда брать данные из нее, если нет – то брать из кэша. Ниже реализация:

**const** CACHE = 'network-or-cache-v1';  
**const** timeout = 400;  
// При установке воркера мы должны закешировать часть данных (статику).  
self.addEventListener('install', (event) => {  
 event.waitUntil(  
 caches.open(CACHE).then((cache) => cache.addAll([  
 '/img/background'  
 ])  
 ));  
});  
  
// при событии fetch, мы и делаем запрос, но используем кэш, только после истечения timeout.  
self.addEventListener('fetch', (event) => {  
 event.respondWith(fromNetwork(event.request, timeout)  
 .catch((err) => {  
 console.log(`Error: ${err.message()}`);  
 **return** fromCache(event.request);  
 }));  
});  
  
// Временно-ограниченный запрос.  
**function** fromNetwork(request, timeout) {  
 **return** **new** Promise((fulfill, reject) => {  
 **var** timeoutId = setTimeout(reject, timeout);  
 fetch(request).then((response) => {  
 clearTimeout(timeoutId);  
 fulfill(response);  
 }, reject);  
 });  
}  
  
**function** fromCache(request) {  
// Открываем наше хранилище кэша (CacheStorage API), выполняем поиск запрошенного ресурса.  
// Обратите внимание, что в случае отсутствия соответствия значения Promise выполнится успешно, но со значением `undefined`  
 **return** caches.open(CACHE).then((cache) =>  
 cache.match(request).then((matching) =>  
 matching || Promise.reject('no-match')  
 ));  
}