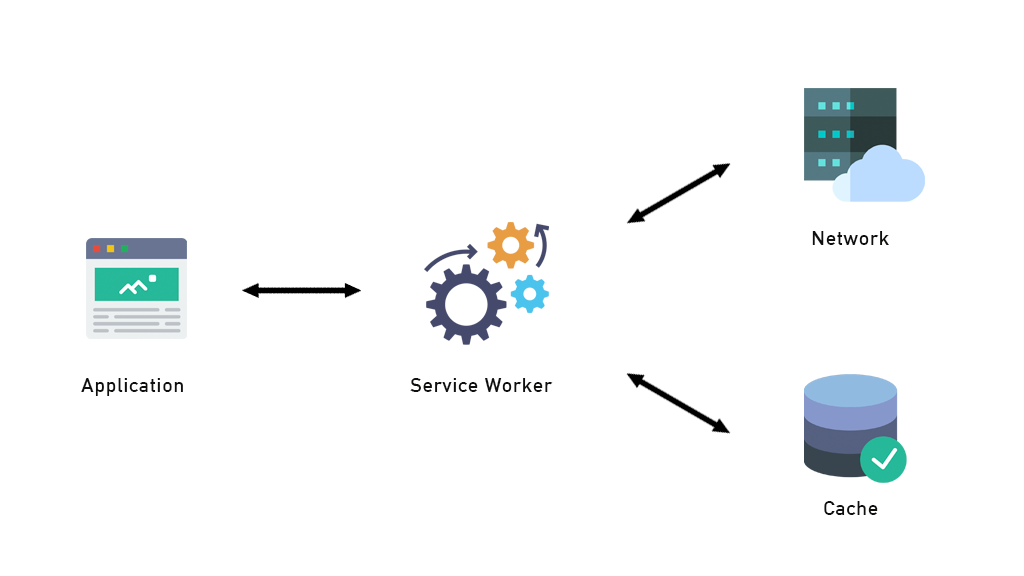
Web App: What is Service Workers?

Service Worker is a script written in Javascript. It works in the background of the browser. To use Service worker, HTTPS must be installed on the server where the application is located.Works through the browser (supported by Chrome, Firefox 4, Opera5 and Microsoft Edge 6),



What can we do with the Service Worker?

It has push notification and background synchronization features.

Push notifications can be sent to users using your application via the browser.

Background synchronization records transactions made in cases where there is no internet connection and transmits these transactions to the server when the connection is made.

Other points you need to know about the service worker are;

It allows you to develop offline applications using the browser's cache API.

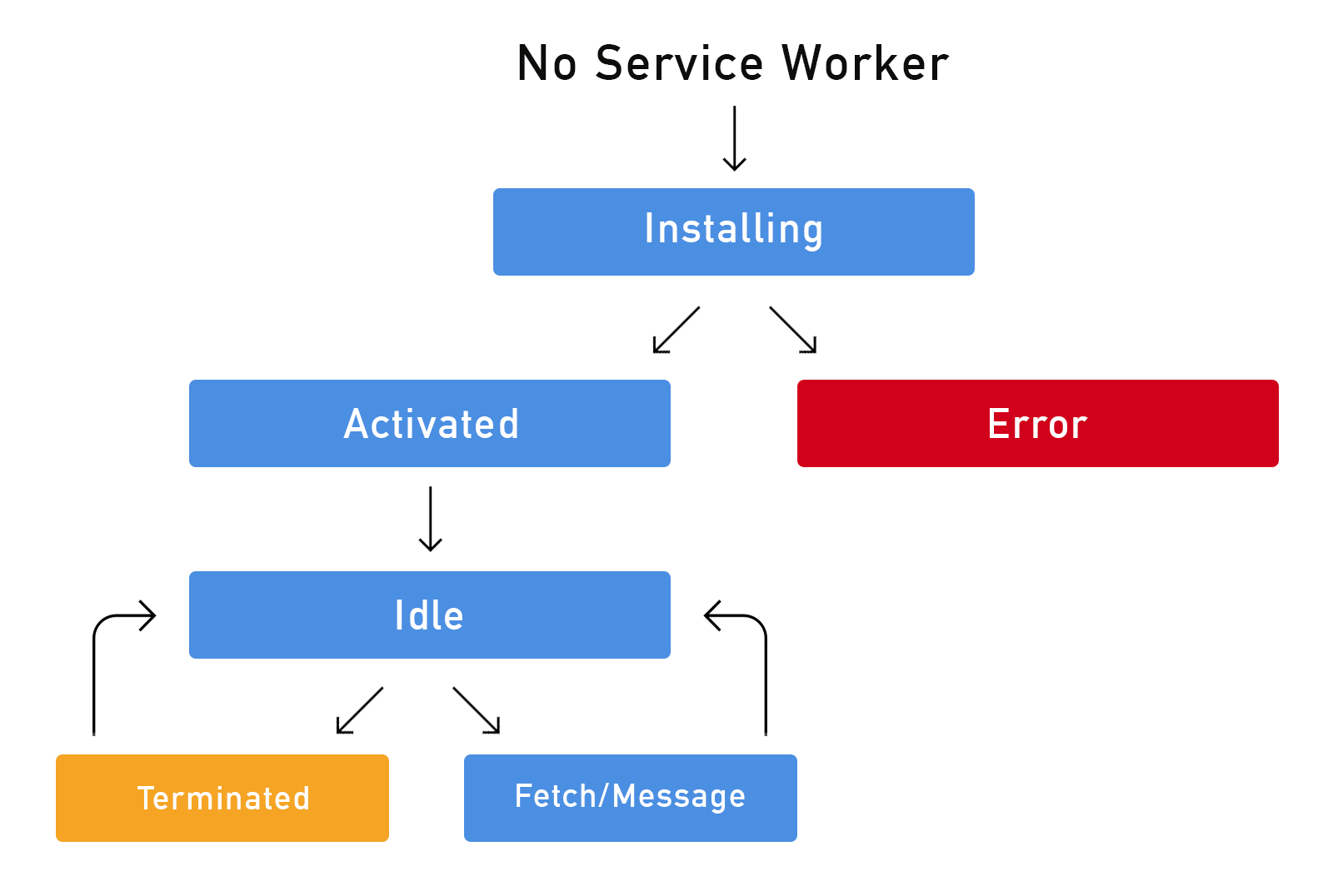
It works like a proxy and allows you to control requests over the network.

It cannot directly access the DOM. Instead, it communicates via postMessage.

It works over the HTTPS protocol.

How does service worker work?

It has a completely separate life cycle from the service worker web application. Let's examine the steps in the life cycle.



1. Register

In order to use the service worker in our application, we must first perform the registration process.

if ('serviceWorker' in navigator) {  
navigator.serviceWorker.register('/sw.js').then((response) => {  
console.log('ServiceWorker registration successful');  
}, (err) => {  
console.log('ServiceWorker registration failed');  
});  
}

In the example above, we check whether our browser supports service workers. If there is browser support, we register our sw.js file as a service worker. We will now manage the processes related to the service worker from this file.

2. Install

In this step, a cache is opened and saved for the data we decide to cache. In the absence of an internet connection, transactions are continued with data cached by the browser.

var cacheName = 'cache-v1';  
var cacheFiles = ['/','/css/main.css','/js/main.js'];  
  
self.addEventListener('install', (event) => {  
event.waitUntil(  
caches.open(cacheName).then((cache) => {  
console.log('Opened cache');  
return cache.addAll(cacheFiles);  
})  
);  
});

In the example above, the name and data are determined for the cache and the registration process is performed. Once the installation process is completed, if the service worker changes, it is run again.

In this example, we see that we can manage requests over the network with the fetch event. We can block or redirect data that we have cached.

self.addEventListener('fetch', (event) => {  
event.respondWith(  
caches.match(event.request).then((response) => {  
if (response) return response;  
return fetch(event.request);  
}  
)  
);  
});

3.Activate

In this step, the service worker is active. If no action is performed on the service worker, it switches to the "Idle" state.

self.addEventListener("activate", (event) => {   
console.log("Activating Service Worker...");   
return self.clients.claim();   
});

To sum up;

Service worker ensures that you can communicate with users with its push notification feature and your application is fast and accessible by preserving the user experience with its cache mechanism.

My next week: adding screen sharing to the system

Estimated duration: 3 hours