Service Workers

The service workers API it's a very hot topic in web development. As it can really create good experiences on the web for progressive web apps, websites, or any kind of web content that you are currently creating for the web platform. Service workers are important today because they can enable new experiences such as making your content available while the user is offline,creating a good performance on any network situations, such if you are in a 2G connection, and it can also improve performance for normalonline operations.

Modern Use cases:

- Offline
- Slow connection
- Unreliable connection
- Progressive web app(PWA)
- Faster web user experiences

Service Workers definition:

- A W3C standard spec of browsers.
- Works on top of the web worker specification
- It manages a scope: a folder in a domain
- It has abilities on that scope
- It works detached from any browser's tab or PWA's process.

Web Worker Specification:

- It's a JavaScript file running on own thread
- Services workers also having their own process.
- It can import other files through a sync API
- It has no access to user interface API.
- No DOM, No window object.

Scopes:

- Scope: Origin + Path
- Origin: Protocol + Host + Port

Abilities:

- See all the requests that pages in the scope are requesting.
- It can respond for those requests, taking response from cache, or fetching them.
- It can respond for those requests, creating responses on the fly.
- It can receive messages from server when the web app is closed.

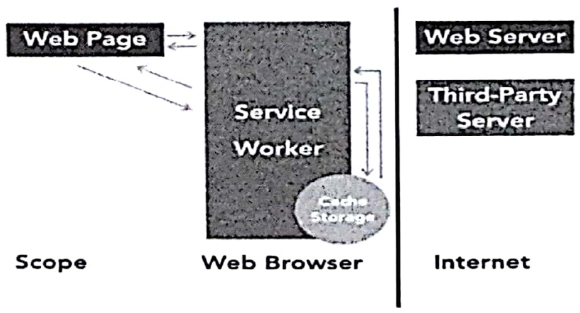
- It can receive payment results from a third-party provider even if the user closed the web app.
- It can synchronize data in the back ground while offline when the connection is reestablished.

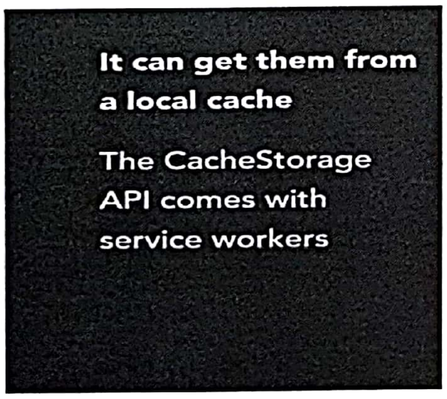
What can we do with a service worker? If we are creating a Progressive Web App, or a PWA, a service worker is mandatory. A service worker is like the brain of a Progressive Web App. So if we want to create these kind of apps we need to create a service worker. In this case the service worker will be responsible for installing all of the apps assets such as CSS, JavaScript, SVG files, images, videos and web phones.

And it's also going to be responsible for serving those files from a local cache. Maybe we don't want to create the Progressive Web App but we want to offer the user an offline experience. In that case, the service worker, will be responsible for that behavior. We can make the whole website available offline, or just part of the website. It's up to us. When we are dealing with web performance, the service worker is a key piece to increase the experience of the first look because the service worker can cache files and serve those files from the cache when the user is trying to access the website again later.

Sometimes, there is connection, the user is online but the server is down, or there is some weird situation. For example, the server is overloaded, or somehow the server is answering with a 404. So the file is not there, so abnormal situations. There service worker can detect and make decisions. Clients app because remember the service worker, if installed, clients app. We can also detect high-latency networksor even captive portals.

For example, you're in a hotel or your in a coffee store, you connect to the wifi but you didn't accept the terms, or you don't have password, so the OS is exposing a network but it is not working. Well, the service worker can detect that situation and do something with that, or replace the server, or act like the server to provide our website or web app the right files. The service worker can also generate local content and that content will be HTTP responses that are not real, so they didn't come from the server.





It can get them from a local cache

The CacheStorage API comes with service workers

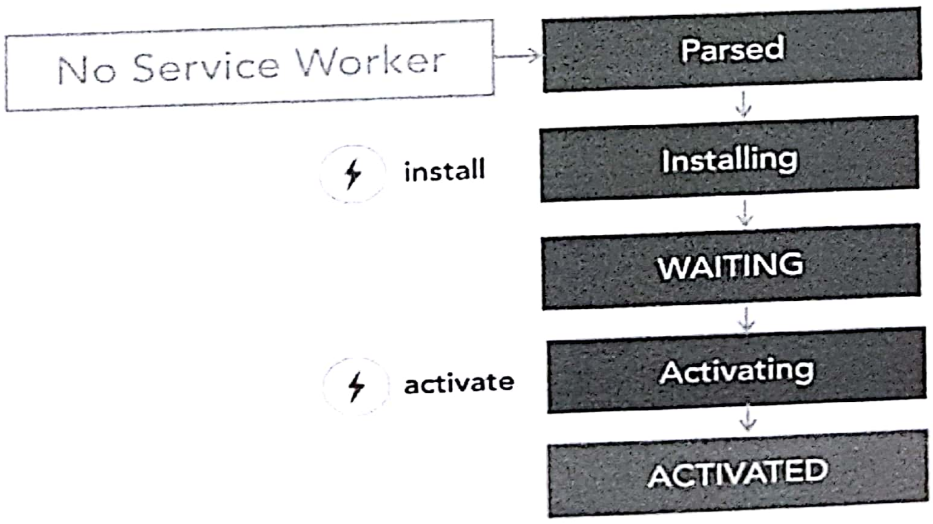
The service worker is acting like a proxy, it's in the middle between the network and the webpage, or the browser, and every request going to the server is trapped, or might be trapped by the service worker. At that point, the service worker can make some decisions.

For example, it can finally fetch those resources from the network, but it's still in the middle. It can also get them from a local cache, very same API known as CacheStorage available with service worker where we can store and retrieve actually be responses. So the service worker can take responses from the cache and deliver those responses to the webpage. And from a webpage point of view, those resources are really coming from the web server.

So from a webpage point of view, you don't know that the service worker was in the middle. Well, you're thinking, who is filling that CacheStorage with the actual responses? Typically the service worker downloaded all the files beforehand when the service worker is being installed. So there is an event, the install event, and that that event, the service worker will download and cache all the resources necessary for later.

The service worker can also synthesize responses so basically it can create responses on the fly that are not in the cache or not coming from a server. In this case we can do this only for same-origin requests. So we cannot fake third party resources such as JavaScript files. And finally, there is this message in API available between the service worker and webpages, PWA's, and iframes, meaning that we can not only drop http request, but also sending and receiving messages from the webpages.

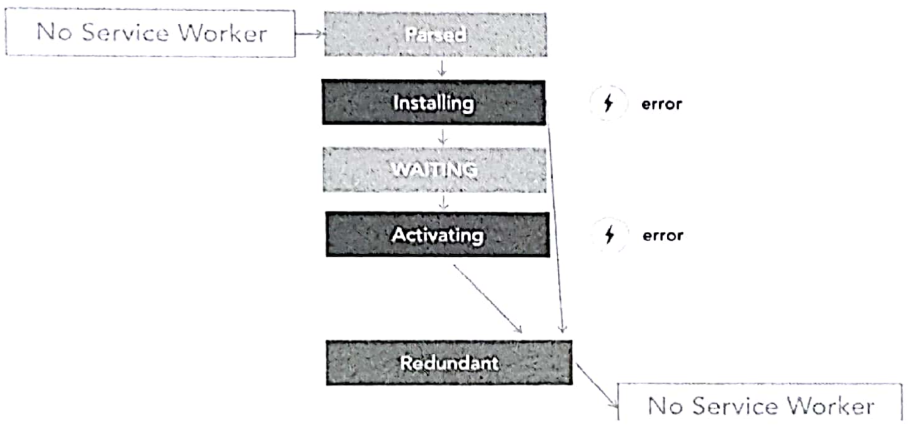
Life cycle of service worker:



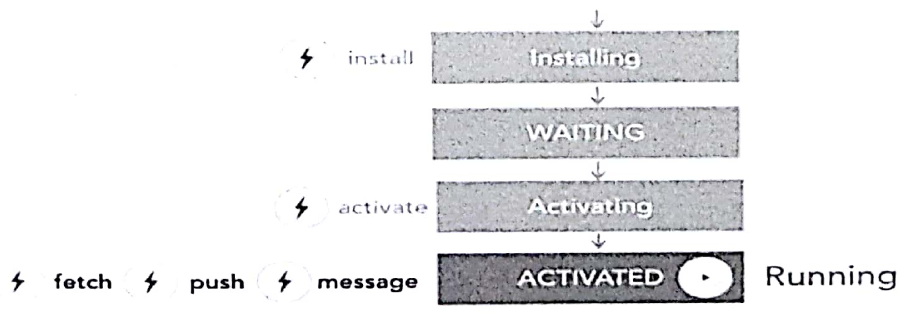
Understanding the life cycle of a service worker is really important because this is new to the web and the life cycle iscompletely different to any child script that you have coded before. So, first time you access a website, there is no service worker installed,and the website is registering that service worker. So, from a service worker's point of view this is what's going on. First, the service worker is parsed, so the file is downloaded by the browser and parsed, and then it go through an installing process, and there is an event that we combine to known as install.

After it was installed, it's waiting. Maybe you're thinking, waiting for what. Well, it's waiting because sometimes there is a previous service worker in memory, so it's waiting for that previous service worker to stop working. After that's happening, we go through the next step that is activation. So, there is an activate event that we can also listen, and after that the service worker is activated. Activated means that it's right now the owner of that scope, so anything that is happening in the scope of that service worker is now the responsibility of this service worker.

But, don't think that activated means that it's currently running because we have something else known as a running status.



If there is any error inside installing or activating, so you are downloading files and that's not working, an error event is fired and after that there is a special state known as redundant that means you're still in memory but you're not really the service worker for that scope, so the next step will bethat you will be removed from user's device.



So, after the service worker is activated, we have a running status. So, it is start as idle, that means that it's not executing any code. There are a series of events that will fire a running status, which means now your service worker is executing code. These events are fetch, push, or message.

chrome://serviceworker-internals. When you get there, you will see all the service workers that are currently installed in your system.

```
//server.js
var express = require("express");
var path = require("path");
var bodyParser = require("body-parser");
var app = express();

console.log(__dirname);
app.use(express.static(path.join(__dirname, "./views")));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.listen(4200);
console.log("server is listening on port 4200");
```

```
//index.html
<script>
   if("serviceWorker" in navigator){
       navigator.serviceWorker.register("serviceWorker.js",{scope:"/"})
       .then(registration=>{
           document.querySelector("#output").innerHTML = "service worker
installed propelly"
       })
       .catch(error=>{
           document.querySelector("#output").innerHTML = "some error in service
workers";
       })
   }
   else{
       document.querySelector("#output").innerHTML = "service workers not
available";
   }
</script>
<div id="output"></div>
```

```
//serviceWorker.js
console.log("we are in service worker");

self.addEventListener("install",event=>{
    console.log("sw is installed");
});
self.addEventListener("activate",event=>{
    console.log("in activate method");
})
```

Import External files:

```
//serviceWorker.js
console.log("we are in service worker");

importScripts("events.js");
//this is synchronous API means if
console.log("after bload scripts");
//line by line execution
```

```
//events.js
self.addEventListener("install",event=>{
    console.log("sw is installed");
});
self.addEventListener("activate",event=>{
    console.log("in activate method");
})
```

Work with registration:

```
<script>
   if("serviceWorker" in navigator){
       navigator.serviceWorker.register("serviceWorker.js",{scope:"/"})
       .then(registration=>{
           scope = registration.scope;
           document.querySelector("#output").innerHTML = `service worker
installed propelly ${scope}`
       })
       .catch(error=>{
           document.querySelector("#output").innerHTML = "some error in service
workers";
       })
   }
   else{
       document.querySelector("#output").innerHTML = "service workers not
available";
   }

//another way of register
   if("serviceWorker" in navigator){
       navigator.serviceWorker.getRegistration()
        .then(registration=>{
            registration.addEventListener("updatefound",event=>{
                const swInstalling = registration.installing;
                swInstalling.addEventListener("statechange",()=>{
                    if(swInstalling.state="installed"){

                    }
                })
            })
        })
        .catch(error=>{

        })
   }

</script>

<div id="output">

</div>
```

Manage updates in our code:

```
//index.html
<script>
    function update(){
        navigator.serviceWorker.getRegistration()
            .then(registration=>{
                registration.update();
            })
        }
    if("serviceWorker" in navigator){
        navigator.serviceWorker.register("serviceWorker.js",{scope:"/"})
        .then(registration=>{
            scope = registration.scope;
            document.querySelector("output").innerHTML = `service worker installed
propelly ${scope}`
        })
        .catch(error=>{
            document.querySelector("output").innerHTML = "some error in service
workers";
        })
    }
    else{
        document.querySelector("output").innerHTML = "service workers not
available";
    }

//another way of register
    if("serviceWorker" in navigator){
        navigator.serviceWorker.getRegistration()
         .then(registration=>{
             registration.addEventListener("updatefound",event=>{
                 const swInstalling = registration.installing;
                 console.log("newwww")
                 swInstalling.addEventListener("statechange",()=>{
                     if(swInstalling.state="installed"){
                         document.querySelector("output").innerHTML = "new service
worker installed and wiaitng for activate";
                     }
                     else{
                         document.querySelector("output").innerHTML = "service
worker is controlled";
                     }
                 })
             })
         })
         .catch(error=>{

         })
    }

</script>
<button onclick="update()">update here</button>
<output></output>
```

```
//serviceWorker.js
//version 2
console.log("we are in service worker");

try{
    importScripts("events.js");
} catch(e){

}

//this is synchronous API means if
console.log("after bload scripts");
//line by line execution
```

Unregister a service worker:

//index.html

```
<button onclick="unregister()">unregister</button>
function unregister(){
        navigator.serviceWorker.getRegistration()
            .then(registration=>{
                registration.unregister();
            })
        }
```

Capture the fetch event

```
//serviceWorker.js
//version 4
console.log("we are in service worker");

try{
    importScripts("events.js");
} catch(e){

}

self.addEventListener("fetch",event=>{
    console.log(`fetch event ${event.request.url}`);
    const response = new Response(`fetch event ${event.request.url}`);
    event.respondWith(response);
})

//this is synchronous API means if
console.log("after bload scripts");
//line by line execution
```