

Experiment No. 8

Aim : To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA.

Theory :

Service Worker

Service Worker is a powerful script that operates in the background of a browser, independently of user interaction. It acts as a network proxy, intercepting outgoing HTTP requests made by your web application. With Service Worker, you can manage network traffic, handle push notifications, and develop "offline first" web applications using the Cache API.

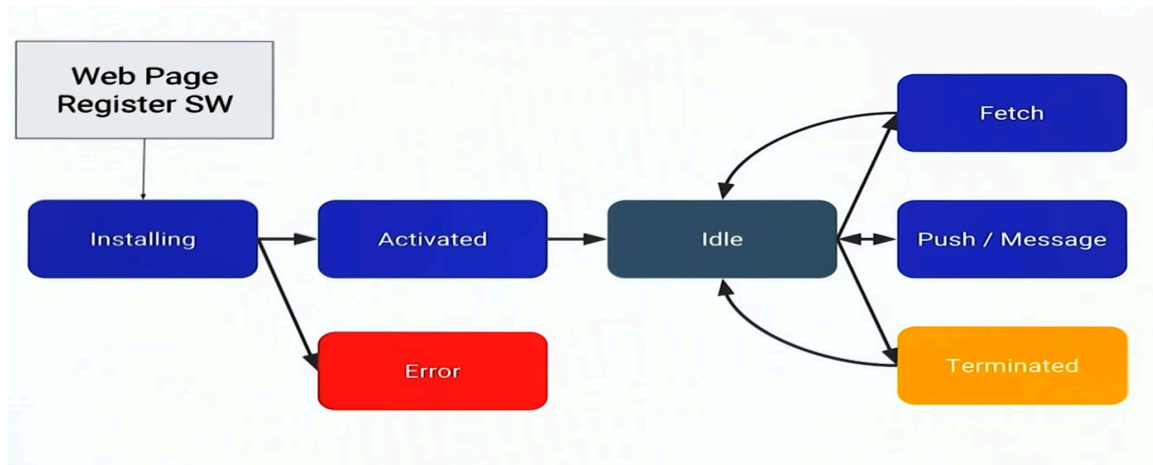
- **Network Proxy:** Service workers intercept all outgoing HTTP requests, allowing you to handle these requests. For example, you can serve content from a local cache if available, improving performance and providing a better user experience.
- **Offline Capabilities:** Service workers enable offline functionality by caching essential application resources such as HTML, CSS, JavaScript, and images. When a user is offline, the service worker can retrieve the requested content from the cache, ensuring a seamless experience even without an internet connection.
- **HTTPS Requirement:** For security reasons, service workers require HTTPS connections. This ensures secure communication between the service worker, your application, and the server.

What can we do with Service Workers?

- **Network Traffic Control:** Manage all network traffic of the page and manipulate requests and responses. For example, you can respond to a CSS file request with plain text or an HTML file request with a PNG file. You can also respond with the requested content.
- **Caching:** Cache any request/response pair with Service Worker and Cache API, allowing you to access offline content anytime.

- **Push Notifications:** Manage push notifications with Service Worker, enabling you to show informational messages to the user.
- **Background Processes:** Even when the internet connection is broken, you can start any process with the Background Sync feature of Service Worker.

Service Worker Cycle



Steps for coding and registering a service worker for your E-commerce PWA completing the install and activation process:

1. Create the Service Worker File (sw.js):

```
JS sw.js > self.addEventListener('install') callback > then() callback
1 self.addEventListener('install', function(event) {
2   event.waitUntil(
3     caches.open('offline')
4     .then(function(cache) {
5       return cache.addAll([
6         '/',
7         '/index.html',
8       ]);
9     })
10  );
11 });
12
13 self.addEventListener('fetch', function(event) {
14   event.respondWith(
15     fetch(event.request)
16     .catch(function() {
17       return caches.match(event.request)
18       .then(function(matching) {
19         return matching || caches.match('offline.html');
20       });
21     })
22   );
23 });
```

2. Register the Service Worker:

In your main JavaScript file (e.g., `main.js` or `app.js`), add the following code:

```
async function registerSW() {  
  if ('serviceWorker' in navigator) {  
    try {  
      await navigator  
        .serviceWorker  
        .register('serviceworker.js');  
    }  
    catch (e) {  
      console.log('SW registration failed');  
    }  
  }  
}
```

Output

The screenshot displays a web browser window with a PWA application. The application's interface includes a navigation menu with 'Home', 'Menu', 'Review', 'About', 'Gallery', and 'Order' options. The main content area features a food image and the text 'ess' and 'Introducing Savory Delights: Y Are you ready to awaken your Look no further! Join us on a gastronomic jour companion in the realm of flav'. The browser's developer tools are open, showing the 'Application' tab. The 'Service workers' section indicates that a service worker is registered at 'http://127.0.0.1:5500'. The 'Storage' section shows a table of cache entries:

#	Name	Response-Type	Content-Type	Content-Length	Time Cached	Vary Header
0	/	basic	text/html	31,730	22/3/2024, 9:54...	Origin
1	/index.html	basic	text/html	31,730	22/3/2024, 9:54...	Origin

The 'Background services' section lists various capabilities like 'Back/forward cache', 'Background fetch', 'Background sync', 'Bounce tracking mitigati', 'Notifications', 'Payment handler', 'Periodic background syn', 'Speculative loads', 'Push messaging', and 'Reporting API'. The 'Frames' section shows the 'top' frame. The 'Console' tab is also visible at the bottom.

The screenshot displays the Chrome DevTools Application panel. The left sidebar shows the 'Application' tab selected, with a tree view containing 'Manifest', 'Service workers', and 'Storage'. The 'Service workers' section is expanded, showing a list of service workers. The main panel displays the details for the service worker at `http://127.0.0.1:5500/`. It includes controls for 'Offline', 'Update on reload', and 'Bypass for network'. Below these, there are links for 'Network requests', 'Update', and 'Unregister'. The 'Source' is listed as `sw.js`, and the 'Received' time is `17/3/2024, 9:13:52 pm`. The status is '#24 activated and is running', with a 'stop' link. There are input fields for 'Push' (containing 'Test push message from DevTools.'), 'Sync' (containing 'test-tag-from-devtools'), and 'Periodic Sync' (containing 'test-tag-from-devtools'). At the bottom, an 'Update Cycle' table shows the progression of the service worker.

#	Name	Response-Type	Content-Type	Content-Length	Time Cached	Vary Header
0	/	basic	text/html	31,730	22/3/2024, 9:54:...	Origin
1	/index.html	basic	text/html	31,730	22/3/2024, 9:54:...	Origin

Version	Update Activity	Timeline
▶ #24	Install	<div></div>
▶ #24	Wait	<div></div>
▶ #24	Activate	<div></div>

Conclusion : I have understood and successfully registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.