

## **Experiment 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:**

A Service Worker is a JavaScript script that runs in the background, separate from the main browser thread, enabling features like:

- Offline caching (storing assets for offline use).
- Push notifications.
- Background sync (delayed data submission when online).
- Network interception (serving cached responses when offline).

### Role of Service Workers in PWAs

Service Workers are crucial for:

1. Offline Functionality – Caches critical assets (HTML, CSS, JS, images).
2. Performance Optimization – Serves cached content instantly.
3. Reliability – Works even under poor/no network conditions.
4. Background Tasks – Handles sync and notifications without user interaction.

### Service Worker Lifecycle

A Service Worker goes through three key phases:

#### 1. Registration

- The main JavaScript file registers the Service Worker using:

```
if ('serviceWorker' in navigator) {  
  navigator.serviceWorker.register('/sw.js')  
    .then(registration => console.log('SW registered!'))  
    .catch(err => console.log('SW registration failed:', err));  
}
```
- The browser downloads and parses sw.js.

#### 2. Installation

- Triggered when the Service Worker is first registered or updated.
- Used to cache static assets (e.g., via Cache API).
- Example:

```
self.addEventListener('install', (event) => {  
  event.waitUntil(  
    caches.open('v1').then((cache) => {  
      return cache.addAll([
```

```

        '/',
        '/index.html',
        '/styles.css',
        '/app.js',
        '/logo.png'
    ]);
    })
    );
});

```

### 3. Activation

- Runs after installation (or when an old Service Worker is replaced).
- Used to clean up old caches.
- Example:

```

self.addEventListener('activate', (event) => {
    event.waitUntil(
        caches.keys().then((cacheNames) => {
            return Promise.all(
                cacheNames.filter((name) => name !== 'v1')
                    .map((name) => caches.delete(name))
            );
        })
    );
});

```

### Fetch Event Handling

- Intercepts network requests to serve cached responses when offline.
- Example:

```

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

```

### Browser Support & HTTPS Requirement

- Supported in Chrome, Firefox, Edge, Safari (partially).
- Requires HTTPS (except localhost for development).

**Code:****1. Service worker registration**

```
if ('serviceWorker' in navigator) {  
  window.addEventListener('load', () => {  
    navigator.serviceWorker.register('/sw.js')  
      .then(registration => {  
        console.log('ServiceWorker registered with scope:', registration.scope);  
      })  
      .catch(err => {  
        console.error('ServiceWorker registration failed:', err);  
      });  
  });  
}
```

**2. Service worker lifecycle**

```
const CACHE_NAME = 'currency-converter-v2';  
const ASSETS_TO_CACHE = [  
  '/',  
  'index.html',  
  'style.css',  
  'index.js',  
  'country-list.js',  
  'pwa.js',  
  'icons/icon-192x192.png',  
  'icons/icon-512x512.png',  
  'assests/change.png',  
  'https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css',  
  'https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.min.css',  
  'https://fonts.googleapis.com/css2?family=Poppins:wght@100;900&display=swap'  
];
```

```
self.addEventListener('install', (event) => {  
  event.waitUntil(  
    caches.open(CACHE_NAME)  
      .then((cache) => {  
        return cache.addAll(ASSETS_TO_CACHE);  
      })  
  );  
});
```

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

```
// Clone the request
const fetchRequest = event.request.clone();

return fetch(fetchRequest).then(
  (response) => {
    // Check if we received a valid response
    if(!response || response.status !== 200 || response.type !== 'basic') {
      return response;
    }

    // Clone the response
    const responseToCache = response.clone();

    caches.open(CACHE_NAME)
      .then((cache) => {
        cache.put(event.request, responseToCache);
      });

    return response;
  }
);
});

self.addEventListener('activate', (event) => {
  const cacheWhitelist = [CACHE_NAME];
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheWhitelist.indexOf(cacheName) === -1) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

#### Console

```
Live reload enabled. (index):110
ServiceWorker registration successful (index):73
```

## Service worker registered:

The screenshot shows the Chrome DevTools interface with the 'Service workers' tab selected. The left sidebar lists various application features, including 'Service workers', 'Storage', 'Background services', and 'Frames'. The main panel displays the details for a service worker registered at `http://127.0.0.1:5500/`. The source is `sw.js`, and it was received on 4/13/2025 at 10:58:09 AM. The status is '#631 activated and is running', with a 'Stop' button. The clients list shows `http://127.0.0.1:5500/`. The 'Push' field contains 'Test push message from DevTools.' with a 'Push' button. The 'Sync' field contains 'test-tag-from-devtools' with a 'Sync' button. The 'Periodic sync' field also contains 'test-tag-from-devtools' with a 'Periodic sync' button. The 'Update Cycle' section shows a timeline for version #631, with stages: Install, Wait, and Activate. Below this, there is a section for 'Service workers from other origins' with a link to 'See all registrations'.

## Cache:

The screenshot shows the Chrome DevTools 'Cache Storage' panel. The left sidebar is the same as the previous image, but 'Cache storage' is selected. The main panel shows the details for the cache at `http://127.0.0.1:5500`. The origin is `http://127.0.0.1:5500`, the bucket name is 'default', it is not persistent, has a relaxed durability, a quota of 0 B, and no expiration. Below this is a table of cached entries:

#	Name	Respons...	Content...	Content...	Time Ca...	Vary Hea...
0	/	basic	text/html	5,287	4/13/20...	Origin
1	/assests/change.png	basic	image/p...	1,309	4/13/20...	Origin
2	/country-list.js	basic	applicati...	3,043	4/13/20...	Origin
3	/index.html	basic	text/html	5,287	4/13/20...	Origin
4	/index.js	basic	applicati...	2,064	4/13/20...	Origin
5	/style.css	basic	text/css	883	4/13/20...	Origin
6	/npm/bootstrap@5.3.3/dist/css/bootstrap.mi...	cors	text/css	27,432	4/13/20...	Accept-E...
7	/ajax/libs/bootstrap-icons/1.8.1/font/bootstra...	cors	text/css	8,381	4/13/20...	Accept-E...
8	/css2?family=Poppinsital,wght@0,100;0,900&...	cors	text/css	0	4/13/20...	Sec-Fetc...

Below the table, it says 'No cache entry selected' and 'Select a cache entry above to preview'. At the bottom, it says 'Total entries: 9'.

**Github link:** <https://github.com/Rakshit5467/Currency-Converter.git>

**Conclusion:**

This experiment successfully implemented a service worker for the Currency Converter PWA, demonstrating registration, installation (caching critical assets), activation (cleaning old caches), and fetch handling for offline functionality. The service worker lifecycle ensures reliable performance, instant loading, and offline access, key features of progressive web apps. The console logs and cache audits confirmed proper execution across all stages.