# 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

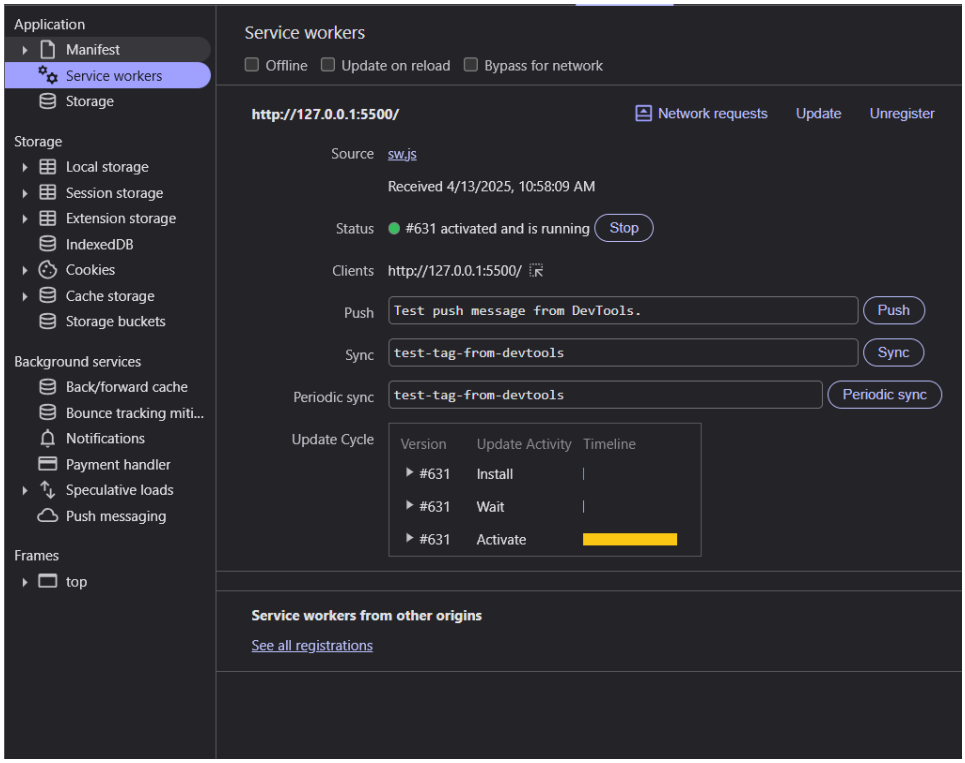
```
const CACHE_NAME = 'currency-converter-v2';
const ASSETS_TO_CACHE = [
  '/',
  '/index.html',
  '/style.css',
  '/index.js',
  '/country-list.js',
  '/pwa.js',
  '/icons/icon-192x192.png',
  '/icons/icon-512x512.png',
  '/assests/change.png',
  'https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css',
  'https://cdnjs.cloudflare.com/ajax/libs/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);
          }
        })
      );
    })
  );
});
```

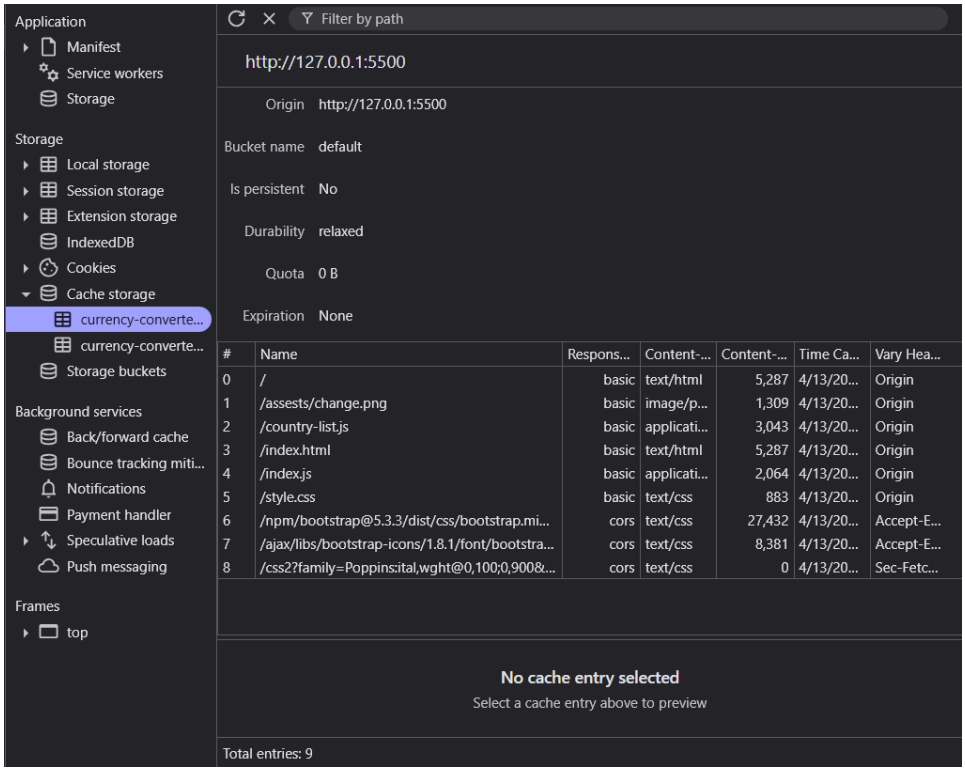
Console

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

## Service worker registered:




## Cache:



**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.