EXPERIMENT NO. 9

Aim: To implement Service worker events like fetch, sync and push for E-commerce PWA.

Theory:
Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop "offline first" web applications with Cache API.

Things to note about Service Worker:

● A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
● Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
● The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.
● Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event
You can track and manage page network traffic with this event. You can check existing cache, manage "cache first" and "network first" requests and return a response that you want.

Of course, you can use many different methods but you can find in the following example a "cache first" and "network first" approach. In this example, if the request's and current location's origin are the same (Static content is requested.), this is called "cacheFirst" but if you request a targeted external URL, this is called "networkFirst".

● **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
● **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

1. `fetch` Event – Handle Resource Requests

```javascript
self.addEventListener('fetch', event => {
  if (event.request.method !== 'GET') return;
  if (event.request.url.includes('/api/')) {
    // Network-first strategy for dynamic API data
    event.respondWith(
      fetch(event.request)
        .then(response => {
          const clone = response.clone();
          caches.open('ecommerce-pwa-v1').then(cache => {
            cache.put(event.request, clone);
          });
          return response;
        })
        .catch(() => caches.match(event.request))
    );
  } else {
    // Cache-first strategy for static assets
    event.respondWith(
      caches.match(event.request).then(cachedResponse => {
        return cachedResponse || fetch(event.request).then(response => {
          const clone = response.clone();
          caches.open('ecommerce-pwa-v1').then(cache => {
            cache.put(event.request, clone);
          });
          return response;
        }).catch(() => {
          if (event.request.headers.get('accept').includes('text/html')) {
            return caches.match('/offline.html');
          }
        });
      })
    );
  }
});
```

2. sync Event – Background Sync for Deferred Requests
Register sync in your app:
```
navigator.serviceWorker.ready.then(reg => {
  return reg.sync.register('sync-cart');
});
```

Handle it in sw.js:
```
self.addEventListener('sync', event => {
  if (event.tag === 'sync-cart') {
    event.waitUntil(syncCartData());
  }
});
function syncCartData() {
  // Example: Sync stored cart data when back online
  return idbKeyval.get('pending-cart').then(cart => {
    if (!cart) return;
    return fetch('/api/sync-cart', {
      method: 'POST',
      body: JSON.stringify(cart),
      headers: {
        'Content-Type': 'application/json'
      }
    }).then(() => {
      // Clear local data after sync
      return idbKeyval.del('pending-cart');
    });
  });
}
```

3. push Event – Handle Push Notifications
Trigger from server:
```
{
  "title": "Order Update",
  "body": "Your order #1234 has shipped!",
  "icon": "/images/icon-192x192.png",
  "badge": "/images/badge.png"
}
```

Handle in sw.js:

```
self.addEventListener('push', event => {
  const data = event.data?.json() || {};
  const title = data.title || "E-commerce Store";
  const options = {
    body: data.body || "Check out our latest updates!",
    icon: data.icon || '/images/icon-192x192.png',
    badge: data.badge || '/images/badge.png'
  };
  event.waitUntil(
    self.registration.showNotification(title, options)
  );
});
```

Handle click on notification:

```
self.addEventListener('notificationclick', event => {
  event.notification.close();
  event.waitUntil(
    clients.openWindow('/orders') // Redirect to orders page or promotional link
  );
});
```