Name : Harsh Jodhav
Roll No: 21
Div : D15B

MPL Experiment No 9.

* Aim :- To implement service worker event like fetch sync for my app.

* Theory :-
Service worker is a script that works on browser background without user interaction independently. Also it reassembles a proxy that works on user guide

- Things to note about service worker :-

- A service worker is a ~~requirement~~ programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only over HTTPS. Because service worker can intercept network requests and modify responses " man in the middle attack could be very bad.

* Fetch Event :-
You can trace and manage page through traffic with this event. You can check existing cache memory caches, manage "cache first" request and restore.

a response that you want.

→ Sync Event :-

Background sync is a web api that is used to delay a process until the internet connection is stable. We can adapt this definition to real life, there is an browser and we want to send an email with this tool.

→ Push Event :-

This is the event that handles push notifications are received from server, you can apply any method with received data.

We can check it by using following methods:

Notification.requestPermission().

* Conclusion:

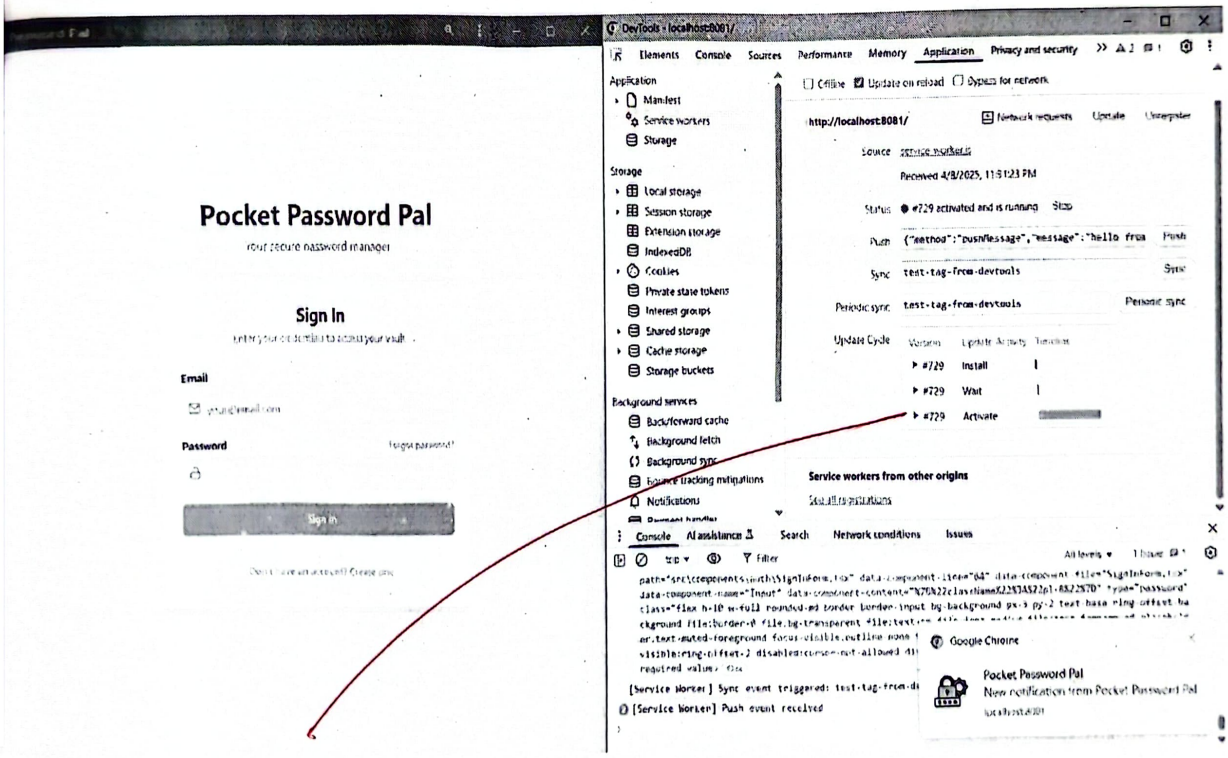
The implementation of fetch, sync push events in a PWA service worker significantly enhances the functionality performance and user experience of modern web application.

Push Event:

```
// Push notification event
self.addEventListener('push', event => {
  console.log('[Service Worker] Push event received');
  const data = event.data ? event.data.json() : {};

  const options = {
    body: data.body || 'New notification from Pocket Password Pal',
    icon: '/icons/icon-192x192.png',
    badge: '/icons/icon-72x72.png',
    data: {
      url: data.url || '/'
    }
  };

  event.waitUntil(
    self.registration.showNotification(data.title || 'Pocket Password Pal', options)
  );
});
```

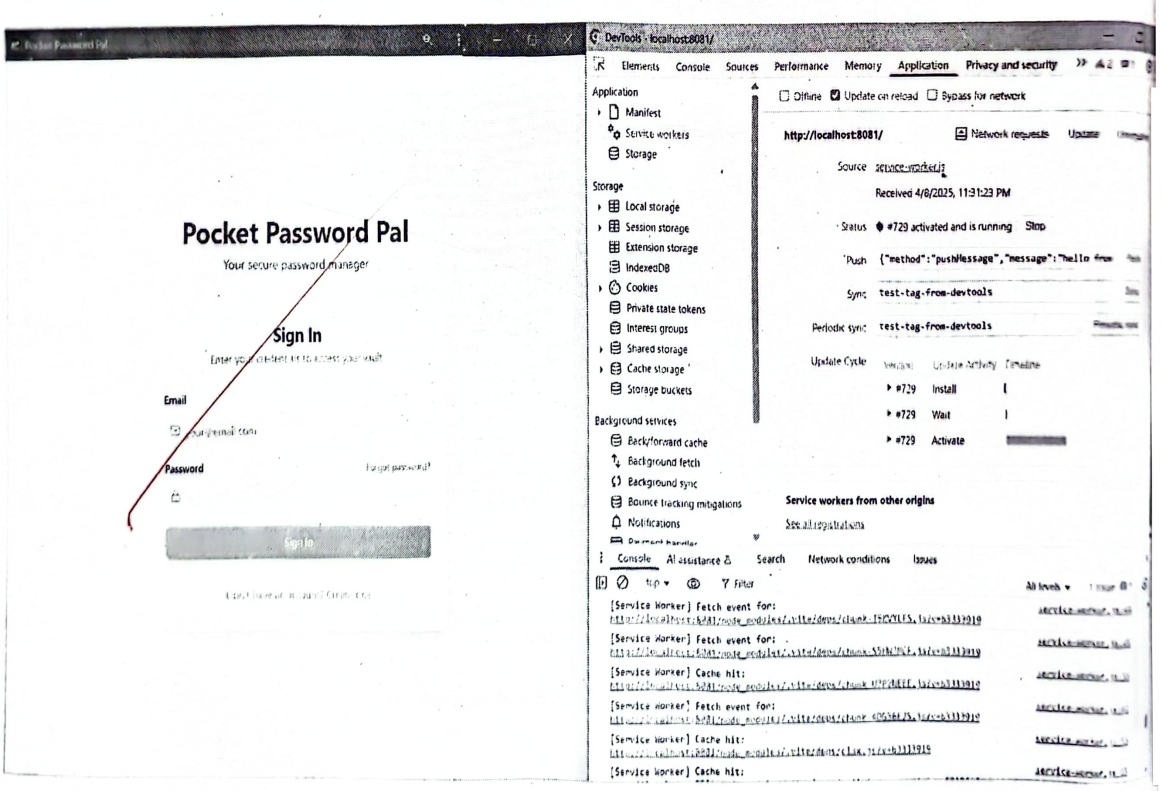


Fetch Event:

```
// Fetch event - respond with cache first, then network
self.addEventListener('fetch', event => {
  console.log(`[Service Worker] Fetch event for: ${event.request.url}`);

  if (event.request.url.includes('supabase.co')) {
    console.log('[Service Worker] Skipping cache for Supabase API call');
```

```
  return;
 }
 event.respondWith(
  caches.match(event.request)
   .then(response => {
    if (response) {
     console.log('[Service Worker] Cache hit:', event.request.url);
     return response;
    }
    console.log('[Service Worker] Cache miss, fetching:', event.request.url);
    return fetch(event.request).then(response => {
     if (!response || response.status !== 200 || response.type !== 'basic') {
      console.log('[Service Worker] Invalid response, not caching');
      return response;
     }
     const responseToCache = response.clone();
     caches.open(CACHE_NAME)
      .then(cache => {
       cache.put(event.request, responseToCache);
       console.log('[Service Worker] Cached new response:', event.request.url);
      });
     return response;
    });
   })
 );
});
```



Sync Event:

```
// Background sync event - for offline data syncing
self.addEventListener('sync', event => {
  console.log(`[Service Worker] Sync event triggered: ${event.tag}`);
  if (event.tag === 'sync-passwords') {
    event.waitUntil(syncPasswords());
  }
});

// Function to sync passwords from IndexedDB to Supabase when online
async function syncPasswords() {
  console.log('[Service Worker] Syncing passwords from offline storage');
  // Your IndexedDB + Supabase logic goes here
}
```