Name: Kshitij Hundre
Div: D15C
Roll no: 18

DOP:
DOS:

Experiment-9

1. Aim

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

2. Basic Description

Progressive Web Apps (PWAs) leverage service workers to provide app-like experiences on the web. Service workers act as programmable proxies that enable features like offline caching, background data sync, and push notifications.

This experiment focuses on three key events supported by service workers:

🔁 1. Fetch Event

- The `fetch` event is triggered whenever the PWA makes a network request.
  It allows the service worker to intercept requests and serve cached responses.
- This improves performance and ensures offline functionality by loading resources from the cache if the network is unavailable.

🔄 2. Sync Event

- The `sync` event enables **background synchronization** when the user is offline.
- Using the `Background Sync API`, we can delay actions (like sending form data or syncing products) until the user is back online.
- This improves reliability in poor or intermittent network conditions.

🔔 3. Push Event

- The `push` event allows the server to push notifications to the user even when the app is not open.
- It's useful for user engagement through marketing messages, order status updates, or reminders.
- The service worker displays the notification using the `showNotification()` method after receiving data from the push server.

Concepts Covered:

- Role and lifecycle of service workers (install, activate, fetch)
- Intercepting and responding to network requests using cache
- Registering and handling background sync tasks
- Receiving and displaying push notifications
- Importance of user permissions for notifications

script.js

```
// Register Service Worker and trigger background sync
if ('serviceWorker' in navigator && 'SyncManager' in window) {
   navigator.serviceWorker.register('sw.js')
    .then((registration) => {
     console.log('✅ Service Worker Registered');

     // Request Notification Permission
     if ('Notification' in window && Notification.permission !== 'granted') {
       Notification.requestPermission().then((permission) => {
         if (permission === 'granted') {
           console.log('🔔 Notification permission granted.');
         } else {
           console.warn('🔕 Notification permission denied.');
         }
       });
     }

     // Trigger background sync
     return registration.sync.register('sync-products');
    })
    .catch((err) => console.error('❌ SW Registration failed:', err));
 }
```

sw.js

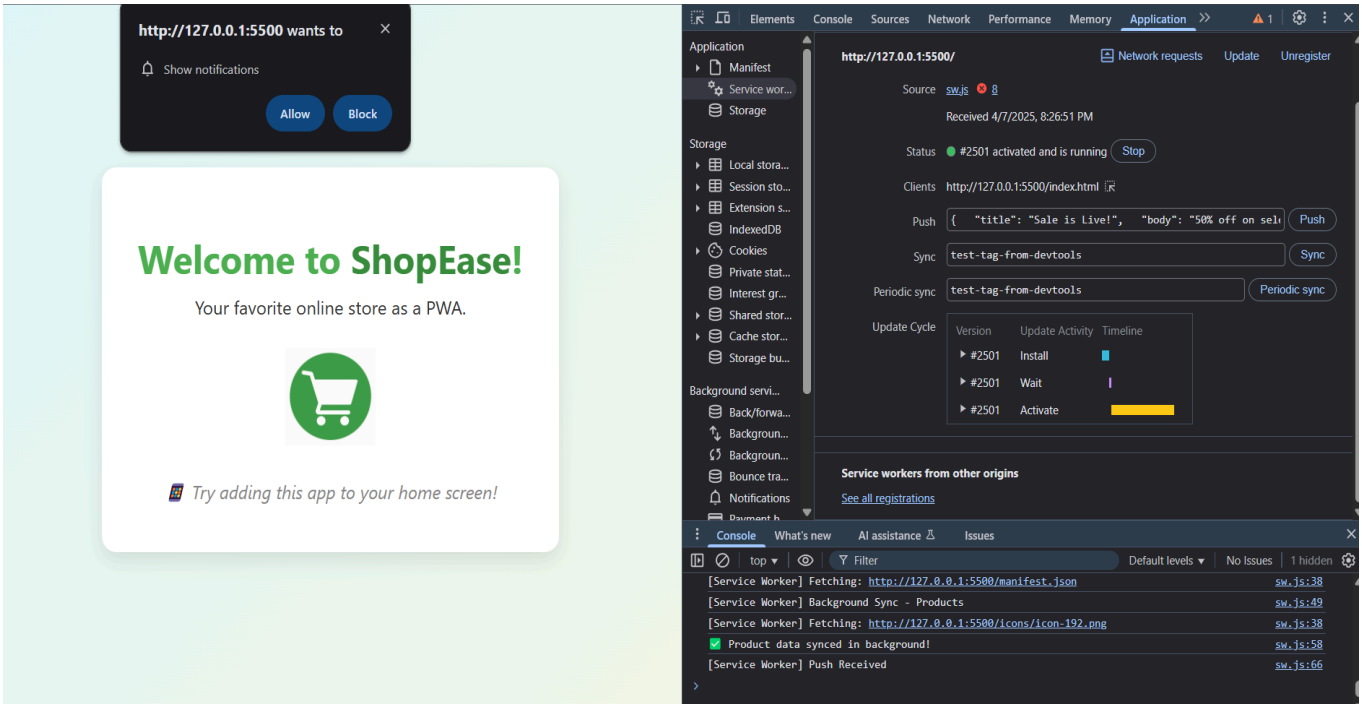
```
// INSTALL: Cache app shell
self.addEventListener('install', (event) => {
   console.log('[Service Worker] Installed');
   event.waitUntil(
    caches.open('v2').then((cache) => {
     return cache.addAll([
       '/',
```

```
        '/index.html',
        '/style.css',
        '/script.js',
        '/manifest.json',
        '/icons/icon-192.png'
      ]);
    })
  );
});

// ACTIVATE
self.addEventListener('activate', (event) => {
  console.log('[Service Worker] Activated');
  // Optional: Clean old caches
  event.waitUntil(
    caches.keys().then((keys) =>
      Promise.all(
        keys.map((key) => {
          if (key !== 'v2') {
            console.log('[Service Worker] Removing old cache:', key);
            return caches.delete(key);
          }
        })
      )
    )
  );
});

// FETCH: Serve from cache or network
self.addEventListener('fetch', (event) => {
  console.log('[Service Worker] Fetching:', event.request.url);
  event.respondWith(
    caches.match(event.request).then((res) => {
      return res || fetch(event.request);
    })
  );
});

// SYNC: Background sync example
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-products') {
    console.log('[Service Worker] Background Sync - Products');
```

```
    event.waitUntil(syncProductData());
  }
});

function syncProductData() {
  return new Promise((resolve) => {
    // Simulated background sync task
    setTimeout(() => {
      console.log('✅ Product data synced in background!');
      resolve();
    }, 2000);
  });
}

// PUSH: Show notification when push received
self.addEventListener('push', (event) => {
  console.log('[Service Worker] Push Received');
  let data = {};

  try {
    data = event.data.json();
  } catch (e) {
    console.warn('Push data is not JSON:', event.data.text());
    data = { title: 'ShopEase Notification', body: event.data.text() };
  }

  const title = data.title || 'ShopEase';
  const options = {
    body: data.body || 'You have a new update!',
    icon: '/icons/icon-192.png',
  };

  event.waitUntil(
    self.registration.showNotification(title, options)
  );
});
```

Github: https://github.com/pixelbypixels/EXP7-PWA.git

3. Output:



4. Conclusion

In this experiment, we successfully implemented three critical service worker events—`fetch`, `sync`, and `push`—to enhance the E-commerce PWA's reliability and engagement. By enabling offline access, background data synchronization, and push messaging, the PWA becomes more user-friendly, robust, and app-like.

This implementation demonstrates how PWAs can provide seamless experiences even in unreliable network conditions, making them a powerful alternative to native mobile apps.