Experiment No.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:**

Service worker:

Service workers play a crucial role in Progressive Web Applications (PWAs) by enabling offline functionality, improving performance, and providing features like push notifications. These scripts run on a separate thread, intercepting network requests, caching resources, and ensuring the application works offline. Service workers need to be registered for a PWA to control network requests, manage caching, and keep the app up to date. They are essential for creating reliable PWAs that can function seamlessly even when the user is offline. Additionally, service workers can handle notifications, perform heavy calculations, and control network requests, making them a powerful tool for enhancing web applications

Service Worker Use:

1. Offline Functionality: Service workers enable PWAs to work offline by caching essential resources, allowing users to access content even when they're not connected to the internet.

2. Improved Performance: By serving cached content locally, service workers enhance performance, reducing server load and speeding up page load times, resulting in a smoother user experience.

3. Push Notifications: Service workers facilitate the delivery of push notifications to users, allowing PWAs to engage users with timely updates and notifications even when the app is not actively in use.

4. Background Sync: Service workers enable background synchronization, allowing PWAs to sync data with servers even when the app is closed or the device is offline, ensuring data consistency and enhancing usability.

5. Network Resilience: Service workers provide network resilience by allowing PWAs to serve content from cached resources in case of network failures, ensuring uninterrupted access to content and enhancing reliability.

Working of service worker:

The service worker is a script that runs in the background of a web application, separate from the main browser thread. Its primary purpose is to handle network requests, intercepting them before they reach the network or the browser's cache. Here's how it works:

1. Registration: The service worker script is registered in the web application's main JavaScript file using the `navigator.serviceWorker.register()` method. Once registered, the service worker is installed and activated.

2. Installation: During installation, the service worker script caches essential resources like HTML, CSS, JavaScript files, and images using the `CacheStorage` API. These cached resources enable the web application to work offline by serving content from the cache when the network is unavailable.

3. Activation: After installation, the service worker enters the activation phase. Here, it takes control of all client pages within its scope and cleans up any outdated caches from previous versions.

4. Interception: Once activated, the service worker listens for various events, such as `fetch`, `push`, and `sync`. When a fetch event occurs (i.e., when the web application makes a network request), the service worker intercepts the request before it reaches the network or the browser's cache.

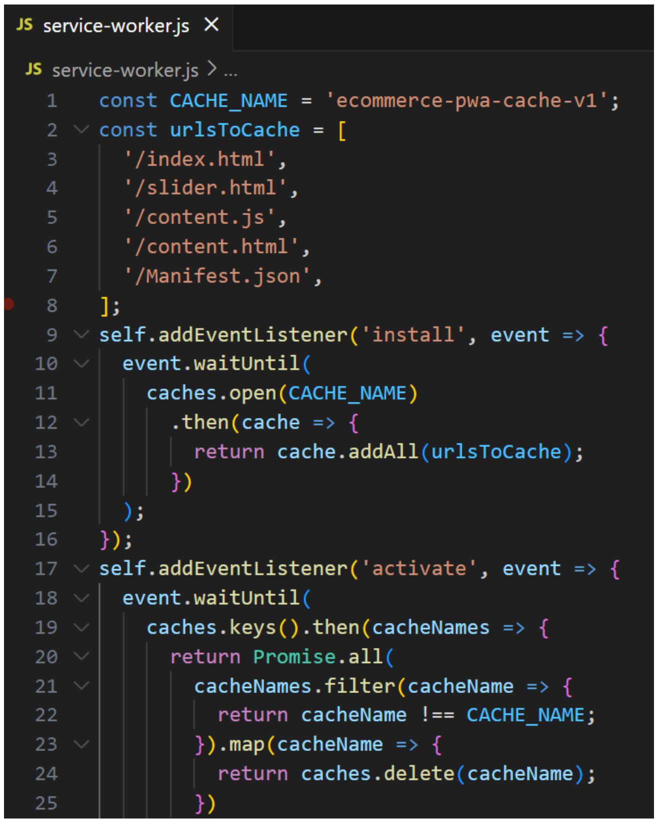
5. Response Handling: The service worker can then respond to the intercepted request in one of several ways:
- If the requested resource is already cached, the service worker retrieves it from the cache and serves it to the web application.
- If the resource is not cached, the service worker fetches it from the network, optionally caching it for future use, and then serves it to the web application.
- The service worker can also intercept responses from the network and modify them before passing them on to the web application.

6. Background Sync and Push Notifications: In addition to intercepting fetch events, service workers can handle other types of events, such as background sync and push notifications. Background sync allows the web application to synchronize data with the server even when the application is not actively in use, while push notifications enable the delivery of timely updates and notifications to users.

- **Code:-**

Create a Service Worker File:

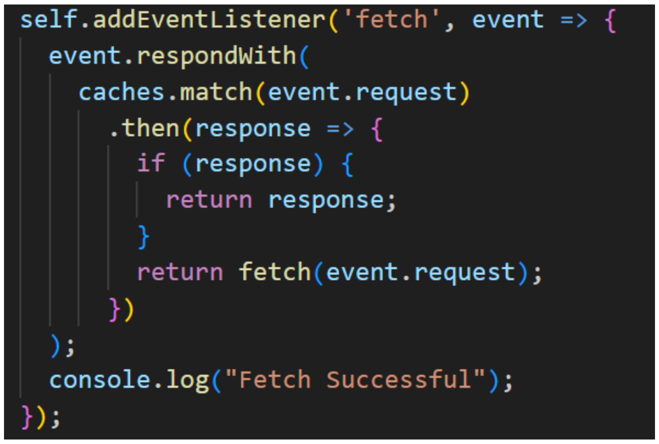
Create a JavaScript file for your service worker. Let's name it service-worker.js. This file will contain the logic for caching assets and handling network requests.



service-worker.js

service-worker.js > ...

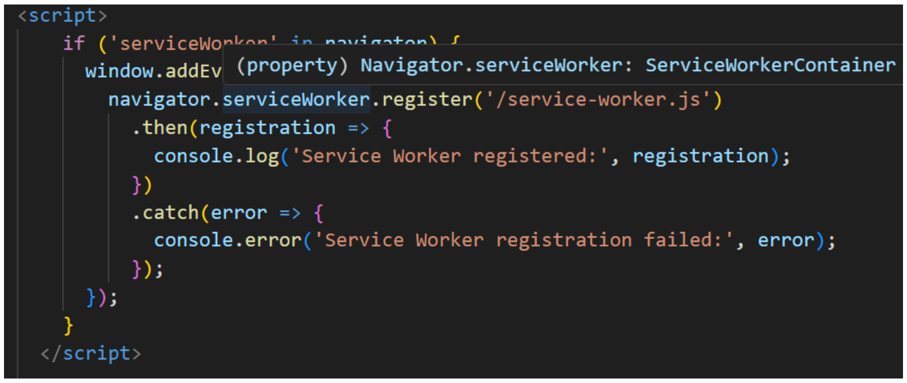
```javascript
const CACHE_NAME = 'ecommerce-pwa-cache-v1';
const urlsToCache = [
  '/index.html',
  '/slider.html',
  '/content.js',
  '/content.html',
  '/Manifest.json',
];
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        return cache.addAll(urlsToCache);
      })
  );
});
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.filter(cacheName => {
          return cacheName !== CACHE_NAME;
        }).map(cacheName => {
          return caches.delete(cacheName);
        })
```



```javascript
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        if (response) {
          return response;
        }
        return fetch(event.request);
      })
  );
  console.log("Fetch Successful");
});
```

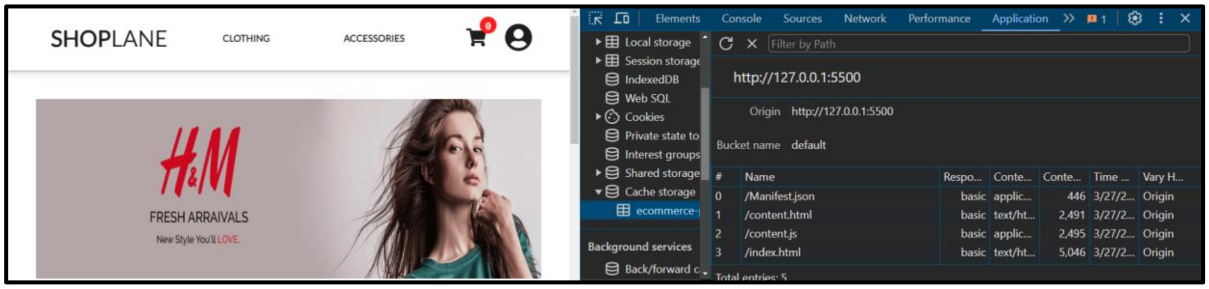
Register the Service Worker (Index.html):

In your main HTML file (e.g., index.html), add a script to register the service worker. Place this script at the end of the <body> tag or just before the closing </body> tag.



```
<script>
    if ('serviceWorker' in navigator) {
      window.addEv (property) Navigator.serviceWorker: ServiceWorkerContainer
        navigator.serviceWorker.register('/service-worker.js')
          .then(registration => {
            console.log('Service Worker registered:', registration);
          })
          .catch(error => {
            console.error('Service Worker registration failed:', error);
          });
      });
    }
  </script>
```

- **Output:-**



Filter by Path

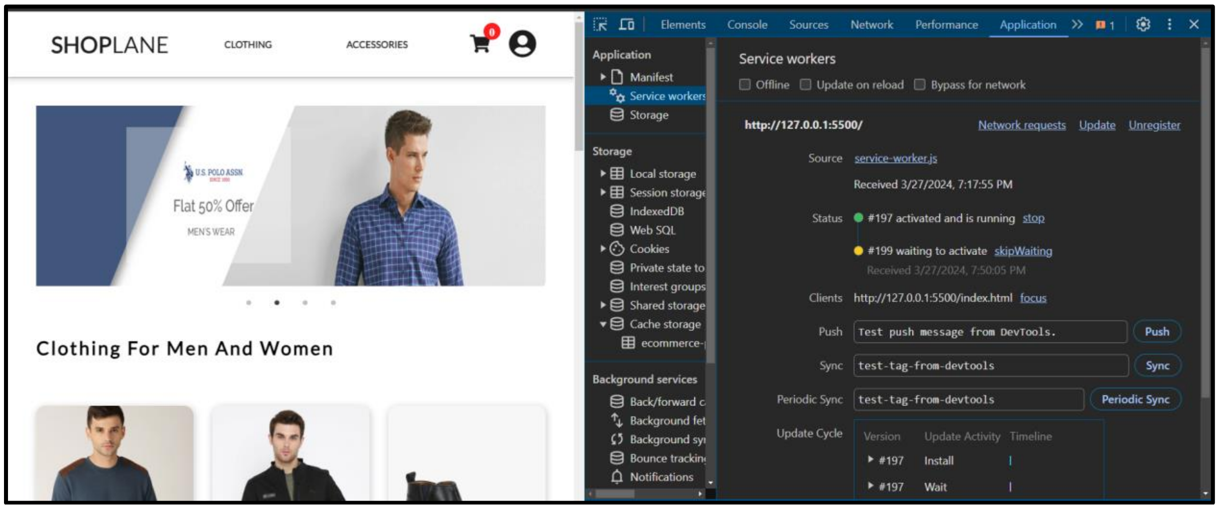
http://127.0.0.1:5500

Origin http://127.0.0.1:5500

Bucket name default

#	Name	Respo...	Conte...	Conte...	Time ...	Vary H...
0	/Manifest.json	basic	applic...	446	3/27/2...	Origin
1	/content.html	basic	text/ht...	2,491	3/27/2...	Origin
2	/content.js	basic	applic...	2,495	3/27/2...	Origin
3	/index.html	basic	text/ht...	5,046	3/27/2...	Origin

Total entries: 5



Service workers

☐ Offline ☐ Update on reload ☐ Bypass for network

http://127.0.0.1:5500/ Network requests Update Unregister

Source service-worker.js

Received 3/27/2024, 7:17:55 PM

Status ● #197 activated and is running stop

● #199 waiting to activate skipWaiting

Received 3/27/2024, 7:50:05 PM

Clients http://127.0.0.1:5500/index.html focus

Push `Test push message from DevTools.` Push

Sync `test-tag-from-devtools` Sync

Periodic Sync `test-tag-from-devtools` Periodic Sync

Update Cycle

Version	Update Activity	Timeline
#197	Install	
#197	Wait	

- **Conclusion:**

In conclusion, by coding and registering a service worker for the E-commerce PWA, we've successfully initiated the process of enabling advanced features such as offline functionality and caching. Completing the install and activation stages ensures that the service worker is ready to intercept network requests and serve cached content, enhancing the user experience by providing seamless access to critical resources even when offline.