PWA Lab
PRACTICAL 8

Name: Mayank Pahuja **Class:D15A Roll no: 40**

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 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.

What can we do with Service Workers?

- You can dominate Network Traffic, manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response.
- You can also send a true response too.
- You can Cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.
- You can manage push notifications with Service Worker and show any information message to the user.
- You can Continue Although Internet connection is broken, you can start any process with Background Sync of Service Worker

What can’t we do with Service Workers?

● You can’t access the window, therefore, You can’t manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.

● You can’t work it on 80 Port
Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle
A service worker goes through three steps in its life cycle:
● Registration
● Installation
● Activation

Registration
To install a service worker, you need to register it in your main JavaScript code. Registration tells the browser where your service worker is located, and to start installing it in the background.

Installation
Once the browser registers a service worker, installation can be attempted. This occurs if the service worker is considered to be new by the browser, either because the site currently doesn't have a registered service worker, or because there is a byte difference between the new service worker and the previously installed one.

A service worker installation triggers an install event in the installing service worker. We can include an install event listener in the service worker to perform some task when the service worker installs. For instance, during the install, service workers can precache parts of a web app so that it loads instantly the next time a user opens it (see caching the application shell). So, after that first load, you're going to benefit from instant repeat loads and your time to interactivity is going to be even better in those cases.

Activation
Once a service worker has successfully installed, it transitions into the activation stage. If there are any open pages controlled by the previous service worker, the new service worker enters a waiting state. The new service worker only activates when there are no longer any pages loaded that are still using the old service worker. This ensures that only one version of the service worker is running at any given time.
When the new service worker activates, an activate event is triggered in the activating service worker. This event listener is a good place to clean up outdated caches

Code & Implementation

New-service-worker code:-

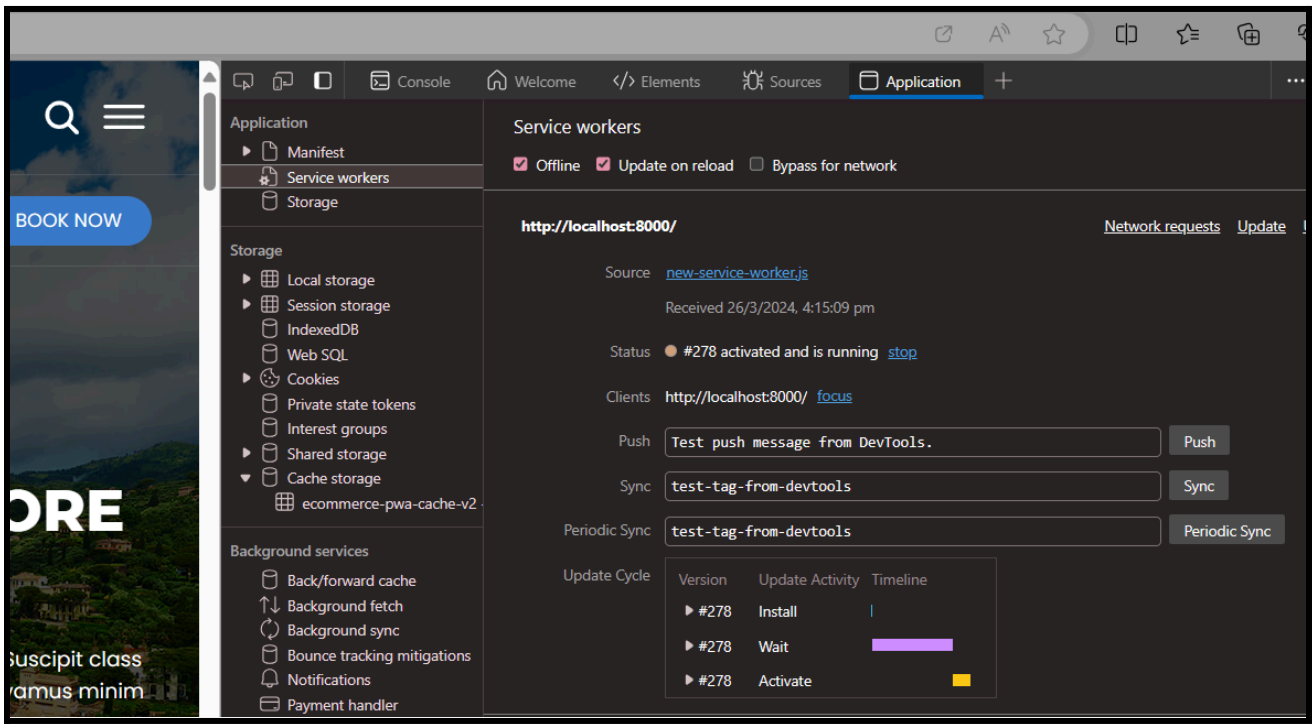
```
const CACHE_NAME = 'ecommerce-pwa-cache-v2';
const urlsToCache = [
    '/',
    '/index.html',
    'assets/css/style.css',
    '/assets/js/script.js',
    'assets/images',
  // Add more URLs of assets to cache as needed
];
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Opened 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);
        }));}));});
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        return response || fetch(event.request);
      }));});
```

Linking the new service worker in html code

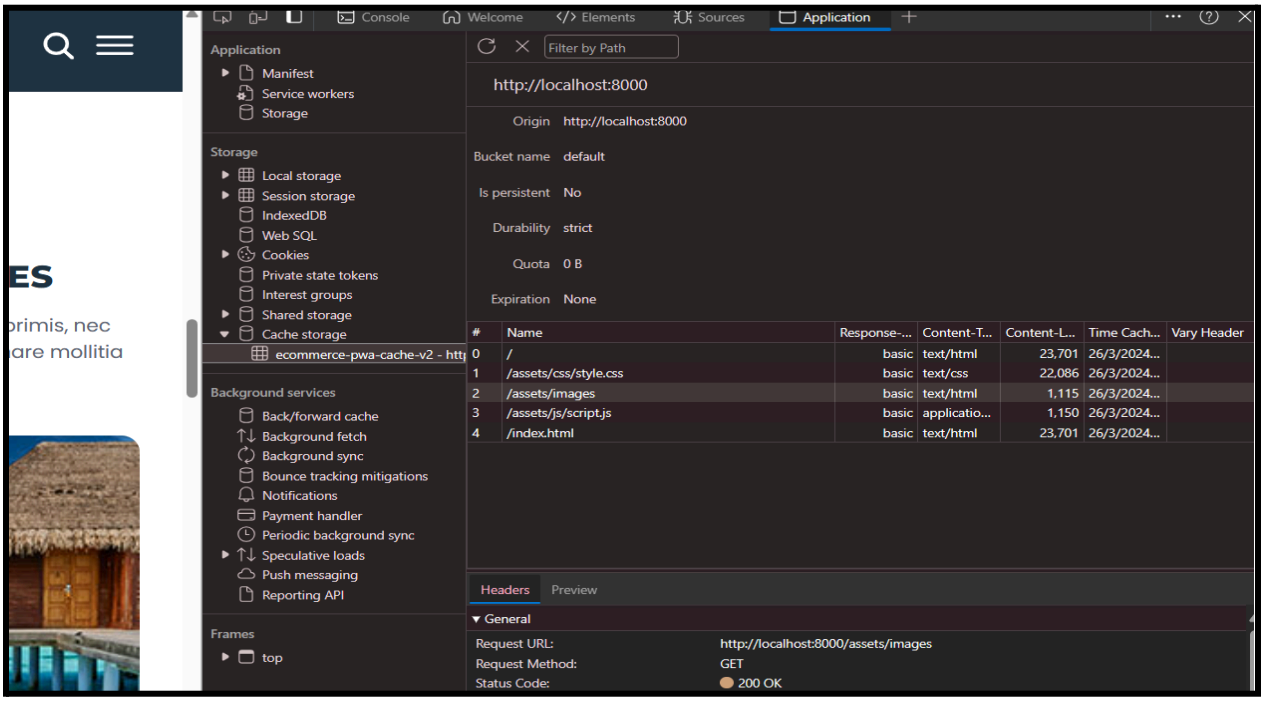
```
<script>
    if ('serviceWorker' in navigator) {
      window.addEventListener('load', () => {
        navigator.serviceWorker.register('/new-service-worker.js')
          .then(registration => {
            console.log('New Service Worker registered with scope:', registration.scope);
          })
          .catch(error => {
            console.error('New Service Worker registration failed:', error);
          });});}
  </script>
```

Output:-

Application → Service worker → we can see the new -service-worker.js is there



Cache Storage - we can see the name of the newly created service worker with status 200 ok



Conclusion: Registered a service worker, and completed the installation and activation process for a new service worker. Also checked the Cache Storage.