PWA Experiment -8

Jai Navani

D15A 30

Theory:-

Understanding Service Workers

A Service Worker is a JavaScript file that runs independently in the background of a browser without direct user interaction. It acts as a proxy, handling network requests, managing push notifications, and enabling offline functionalities using the Cache API.

Key Characteristics of Service Workers:

- They act as a programmable network proxy, allowing control over network requests.
- Service workers function only over HTTPS due to security concerns, preventing potential "man-in-the-middle" attacks.
- They become idle when not in use and restart as needed. To persist data across restarts, IndexedDB can be used.

Capabilities of Service Workers:

- **Network Traffic Management:** Service workers can intercept and manipulate network requests. For example, when a request is made for a CSS file, the service worker can return a plain text response instead.
- **Caching Mechanism:** They store request-response pairs, making offline access possible.
- **Push Notifications:** They enable push notifications, enhancing user engagement.
- **Background Sync:** They allow processes to continue even when there is no active internet connection.

Limitations of Service Workers:

- **No Direct Access to the Window Object:** Service workers cannot manipulate the DOM directly. However, communication with the window is possible through the `postMessage` API.
- **Must Run on HTTPS:** Service workers function only over secure connections, except when being tested on localhost.

Lifecycle of a Service Worker

A service worker undergoes three main stages:

1. **Registration**
2. **Installation**
3. **Activation**

1. Registration

To initialize a service worker, it must be registered in the main JavaScript file. This process informs the browser about the service worker location and initiates its installation.

Example: Registering a Service Worker

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/service-worker.js')
  .then(function(registration) {
    console.log('Service Worker registered successfully with scope:', 
registration.scope);
  })
  .catch(function(error) {
    console.log('Service Worker registration failed:', error);
  });
}
```

This snippet first checks if the browser supports service workers. If it does, `navigator.serviceWorker.register()` is called, returning a promise. Upon successful registration, the scope of the service worker is logged.

Defining the Scope: The scope determines which files the service worker controls. By default, it covers all files in the same directory and its subdirectories.

Setting a Custom Scope:

```
navigator.serviceWorker.register('/service-worker.js', { scope: '/app/' });
```

In this case, the service worker controls all requests under `/app/`, including subdirectories.

If the service worker needs to cover higher-level directories, the `Service-Worker-Allowed` HTTP header must be configured on the server.

2. Installation

After registration, the browser attempts to install the service worker. If the site lacks an existing service worker or if modifications are detected in the new version, the installation process is triggered.

During installation, service workers can cache essential files to enable faster subsequent page loads.

Example: Installation Event Listener

```
self.addEventListener('install', function(event) {
  // Perform installation tasks like caching resources
});
```

3. Activation

Once installed, the service worker moves to the activation phase. If an older version of the service worker is already in use, the new one enters a waiting state until all instances of the old service worker close.

Example: Activation Event Listener

```
self.addEventListener('activate', function(event) {
  // Clean up outdated caches or perform setup tasks
});
```

Once activated, the new service worker takes control of pages within its scope, intercepting network requests and enabling offline functionalities. However, pages loaded before activation remain unaffected until they are refreshed or reopened.

To immediately apply the new service worker to all active pages, `clients.claim()` can be used.

```
self.addEventListener('activate', function(event) {
  event.waitUntil(clients.claim());
});
```

CODE:-

Changes Made in index.html

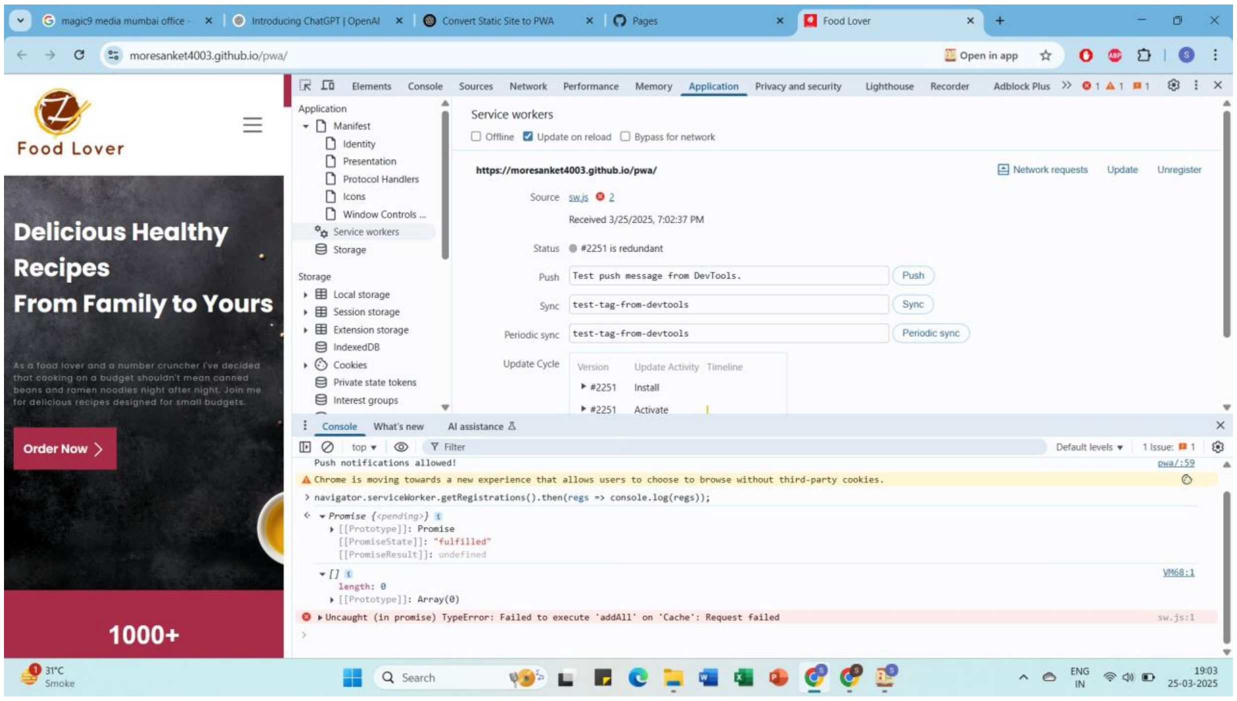
```
<body>
    <script>
        if ("serviceWorker" in navigator) {
          navigator.serviceWorker.register("sw.js")
            .then((reg) => {
              console.log("Service Worker registered!", reg);

              // Background Sync
              if ("SyncManager" in window) {
                navigator.serviceWorker.ready.then((swReg) => {
                  return swReg.sync.register("sync-data");
                });
              }

              // Push Notifications
              if ("PushManager" in window) {
                Notification.requestPermission().then((permission) => {
                  if (permission === "granted") {
                    console.log("Push notifications allowed!");
                  }
                });
              }
            })
            .catch((err) => console.log("Service Worker registration
failed!", err));
        }
      </script>
```

sw.js

```
const CACHE_NAME = "pwa-cache-v1";
const urlsToCache = [
  "/index.html",
  "/assets/css/style.css",   // Update with actual CSS filename
  "/assets/images/icon-192x192.png",
  "/assets/images/icon-512x512.png"
];

// Install event: Caches assets
self.addEventListener("install", (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      return cache.addAll(urlsToCache);
    })
  );
});

// Fetch event: Serve cached files when offline
self.addEventListener("fetch", (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request);
    })
  );
});

// Sync event: Background sync (requires registration in main script)
self.addEventListener("sync", (event) => {
  if (event.tag === "sync-data") {
    event.waitUntil(syncDataFunction());
  }
});
```

```
async function syncDataFunction() {
  console.log("Syncing data in the background...");
  // Example: Send stored requests to server when online
}

// Push event: Handle push notifications
self.addEventListener("push", (event) => {
  const options = {
    body: "New message received!",
    icon: "/assets/images/icon-192x192.png",
    badge: "/assets/images/icon-192x192.png"
  };
  event.waitUntil(
    self.registration.showNotification("PWA Notification", options)
  );
});

// Activate event: Clean up old caches
self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames
          .filter((cacheName) => cacheName !== CACHE_NAME)
          .map((cacheName) => caches.delete(cacheName))
      );
    })
  );
});
```

OUTPUT:-



Service workers

Offline [✓ Update on reload] Bypass for network

https://moresanket4003.github.io/pwa/ — Network requests · Update · Unregister

Source sw.js 2
Received 3/25/2025, 7:02:37 PM

Status #2251 is redundant

Push: Test push message from DevTools.
Sync: test-tag-from-devtools
Periodic sync: test-tag-from-devtools

Update Cycle

Version	Update Activity	Timeline
#2251	Install	
#2251	Activate	

```
Push notifications allowed!                                                          pwa/:59
⚠ Chrome is moving towards a new experience that allows users to choose to browse without third-party cookies.
> navigator.serviceWorker.getRegistrations().then(regs => console.log(regs));
<· ▾Promise {<pending>}
    ▸[[Prototype]]: Promise
     [[PromiseState]]: "fulfilled"
     [[PromiseResult]]: undefined
  ▾[]                                                                                 VM68:1
     length: 0
    ▸[[Prototype]]: Array(0)
✖ ▸Uncaught (in promise) TypeError: Failed to execute 'addAll' on 'Cache': Request failed   sw.js:1
```