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 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
  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
  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
  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.

Code:

```
//Serviceworker.js

// sw.js - Complete Service Worker for E-commerce PWA
const CACHE_NAME = 'ecommerce-pwa-v2';
const API_CACHE = 'ecommerce-api-v1';
const ASSETS_TO_CACHE = [
  '/',
  '/index.html',
  '/manifest.json',
  '/offline.html',
  '/css/main.min.css',
  '/js/app.min.js',
  '/icons/icon-192x192.png',
  '/icons/icon-512x512.png',
  '/images/placeholder-product.jpg'
];

// ========================
// Install Event
// ========================
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('[Service Worker] Cache opened');
        return cache.addAll(ASSETS_TO_CACHE);
      })
      .then(() => self.skipWaiting())
```

```
  );
});

// ========================
// Activate Event
// ========================
self.addEventListener('activate', (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME && cacheName !== API_CACHE) {
            console.log('[Service Worker] Deleting old cache:', cacheName);
            return caches.delete(cacheName);
          }
        })
      );
    })
    .then(() => self.clients.claim())
  );
});

// ========================
// Fetch Event
// ========================
self.addEventListener('fetch', (event) => {
  const { request } = event;
  const url = new URL(request.url);

  // 1. Skip non-GET requests and chrome-extension
  if (request.method !== 'GET' || url.protocol === 'chrome-extension:') {
    return;
  }

  // 2. API Requests (Network First with Cache Fallback)
  if (url.pathname.startsWith('/api/')) {
    event.respondWith(
      fetch(request)
        .then(networkResponse => {
          // Cache successful API responses
```

```
        if (networkResponse.ok) {
          const clone = networkResponse.clone();
          caches.open(API_CACHE)
            .then(cache => cache.put(request, clone));
        }
        return networkResponse;
      })
      .catch(() => {
        // Return cached version if available
        return caches.match(request)
          .then(cachedResponse => cachedResponse || Response.json(
            { error: 'Network error' },
            { status: 503 }
          ));
      })
  );
  return;
}

// 3. Static Assets (Cache First with Network Fallback)
event.respondWith(
  caches.match(request)
    .then(cachedResponse => {
      // Return cached version if found
      if (cachedResponse) {
        return cachedResponse;
      }

      // Otherwise fetch from network
      return fetch(request)
        .then(networkResponse => {
          // Cache successful responses
          if (networkResponse.ok) {
            const clone = networkResponse.clone();
            caches.open(CACHE_NAME)
              .then(cache => cache.put(request, clone));
          }
          return networkResponse;
        })
        .catch(() => {
```

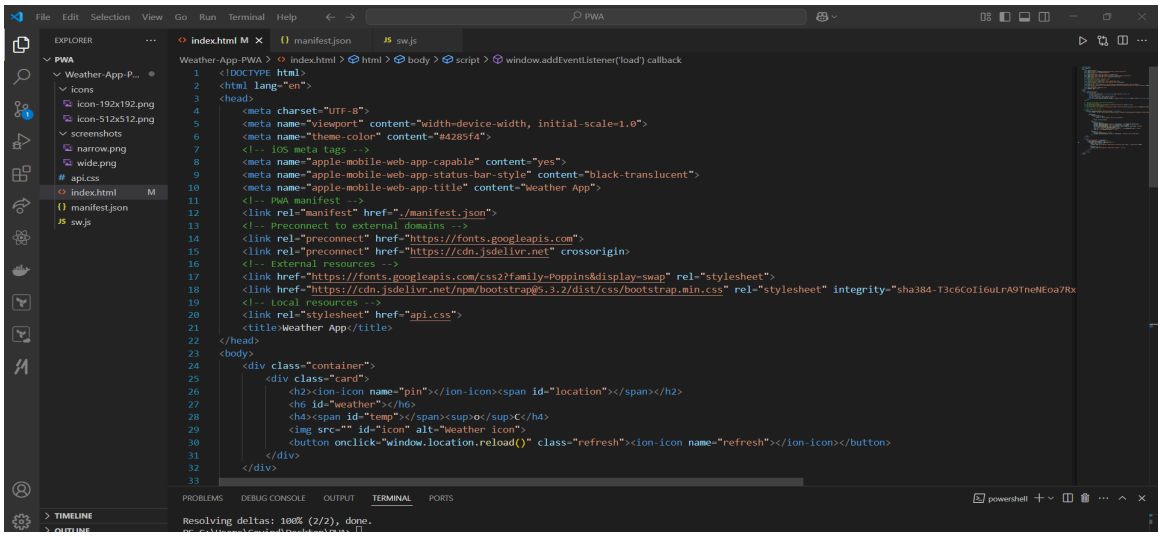
```
          // Special handling for HTML pages
          if (request.headers.get('accept').includes('text/html')) {
            return caches.match('/offline.html');
          }
          // Return placeholder for images
          if (request.headers.get('accept').includes('image')) {
            return caches.match('/images/placeholder-product.jpg');
          }
        });
    })
  );
});

// ========================
// Background Sync
// ========================
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-cart') {
    event.waitUntil(
      // Get cart data from IndexedDB
      getCartData()
        .then(cartItems => {
          return fetch('/api/cart/sync', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(cartItems)
          });
        })
        .then(() => {
          return showNotification('Cart Synced', 'Your cart has been updated');
        })
        .catch(err => {
          console.error('Sync failed:', err);
        })
    );
  }
});

// ========================
// Push Notifications
```

```
// ======================
self.addEventListener('push', (event) => {
  let data = {};
  try {
    data = event.data.json();
  } catch (e) {
    data = {
      title: 'New Update',
      body: 'Check out our latest products!',
      icon: '/icons/icon-192x192.png',
      url: '/'
    };
  }

  const options = {
    body: data.body,
    icon: data.icon || '/icons/icon-192x192.png',
    badge: '/icons/icon-96x96.png',
    data: {
      url: data.url || '/'
    }
  };

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

self.addEventListener('notificationclick', (event) => {
  event.notification.close();
  event.waitUntil(
    clients.matchAll({ type: 'window' })
      .then(clientList => {
        for (const client of clientList) {
          if (client.url === event.notification.data.url && 'focus' in client) {
            return client.focus();
          }
        }
        if (clients.openWindow) {
          return clients.openWindow(event.notification.data.url);
```

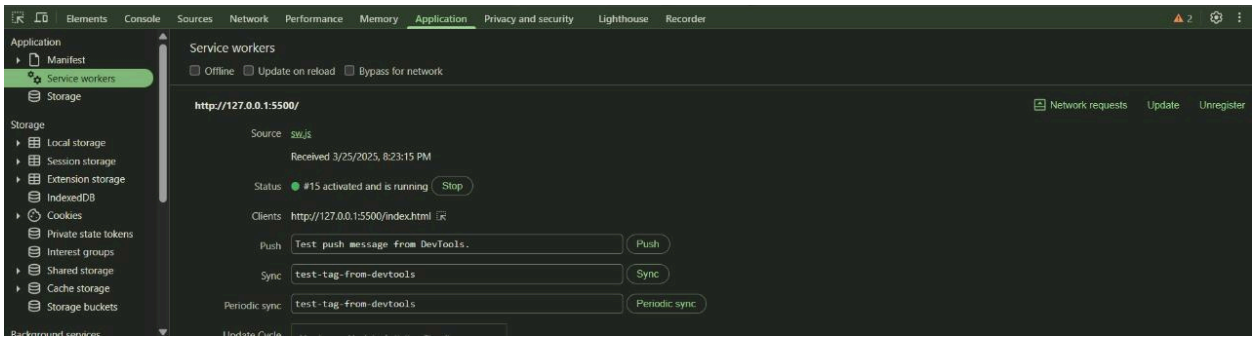
```
      }
    })
  );
});

// ========================
// Helper Functions
// ========================
async function getCartData() {
  // In a real app, you would use IndexedDB
  return new Promise(resolve => {
    resolve([]);
  });
}

async function showNotification(title, body) {
  return self.registration.showNotification(title, { body });
}
```

Output:

Index.html file



Verifying the manifest.json and service workers



Conclusion : Service Workers enable powerful offline capabilities and performance optimizations for web apps but are limited by DOM restrictions and HTTPS requirements. They're essential for building reliable PWAs with features like caching and push notifications