MPL Experiment 9 (PWA)

Name: Devansh Wadhwani - 64

Kunal Punjabi - 43

Manav Punjabi – 44

Hitesh Rohra - 46 **Class:** D15A

Aim: To implement Service worker events like fetch, sync and push for 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.
- Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event

You can track and manage page network traffic with this event. You can check existing cache, manage “cache first” and “network first” requests and return a response that you want.

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”.

- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.

- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

Sync Event

Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn't realize it. When completing the writing, we click the send button.

Push Event

This is the event that handles push notifications that are received from the server. You can apply any method with received data.

We can check in the following example.
"Notification.requestPermission();" is the necessary line to show notification to the user. If you don't want to show any notification, you don't need this line.

In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has "method" and "message" properties. If the method value is "pushMessage", we open the information notification with the "message" property.

Code:

Serviceworker.js

```
const CACHE_NAME = 'delice-cache-v1';
const urlsToCache = [
    './',
    './index.html',
    './styles.css',
    './script.js',
    './icon-192x192.png',
    './icon-512x512.png'
];

// Install Service Worker
self.addEventListener('install', event => {
    event.waitUntil(
        caches.open(CACHE_NAME).then(cache => {
            console.log('Opened cache');
            return cache.addAll(urlsToCache);
        })
    );
});

// Activate Service Worker (Old cache cleanup)
self.addEventListener('activate', event => {
    event.waitUntil(
        caches.keys().then(cacheNames => {
            return Promise.all(
                cacheNames.filter(cache => cache !== CACHE_NAME).map(cache =>
caches.delete(cache))
            );
        })
    );
});

// Fetch Requests with Network Fallback
self.addEventListener('fetch', event => {
    event.respondWith(
        caches.match(event.request).then(response => {
            return response || fetch(event.request)
                .then(networkResponse => {
                    return caches.open(CACHE_NAME).then(cache => {
                        cache.put(event.request, networkResponse.clone());
                        return networkResponse;
                    });
                });
        }).catch(() => {
            return caches.match('./index.html'); // Offline fallback
        })
    );
});
```

Push Notification Code:

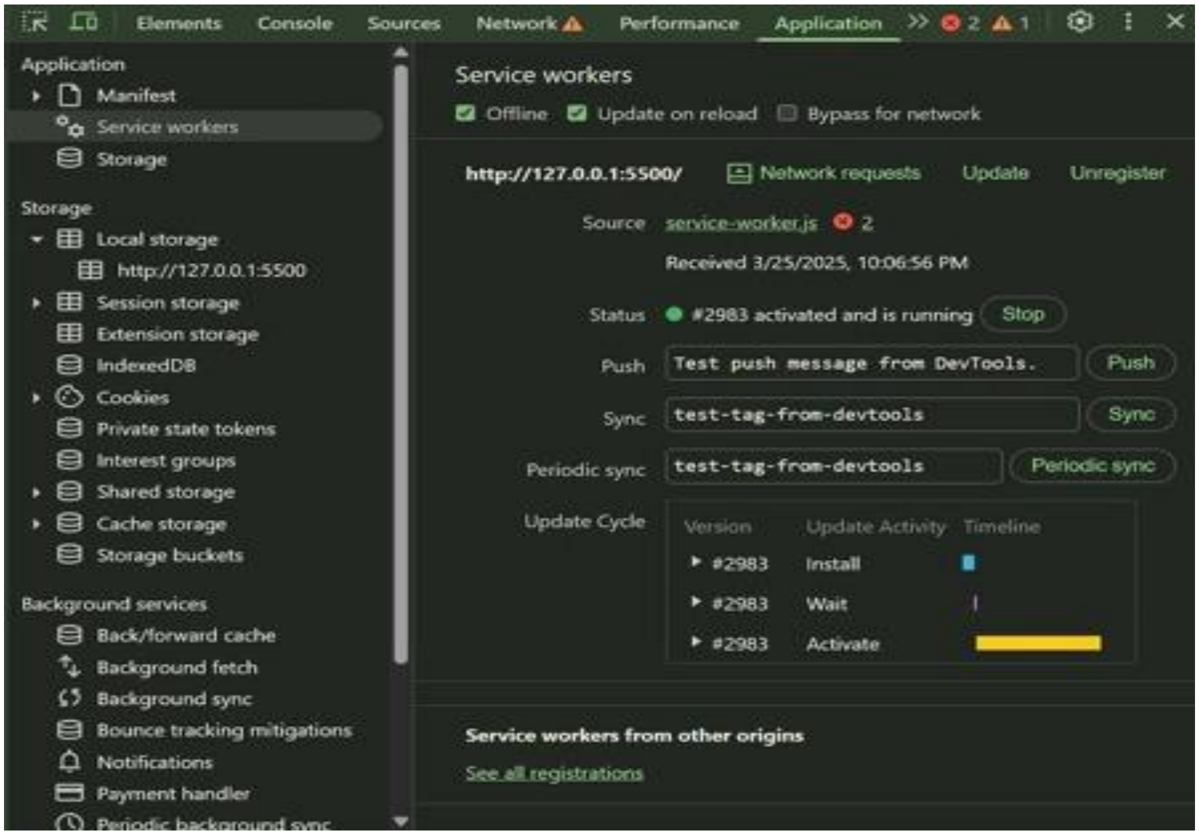
1. Push Notification code:

```
// Push Notification Handling
self.addEventListener("push", (event) => {
   console.log("Push notification received:", event);

   let data = { title: "New Notification", body: "Hello, this is a default message!" };

   if (event.data) {
      try {
         data = event.data.json();
      } catch (error) {
         console.warn("Push message is not JSON, using text instead.");
         data.body = event.data.text();
      }
   }

   const options = {
      body: data.body || "You have a new update!",
      icon: "./assets/images/logo_tourly_192.png",
      badge: "./assets/images/logo_tourly_192.png",
      vibrate: [200, 100, 200],
   };

   event.waitUntil(
      self.registration.showNotification(data.title || "Hello Sannidhi", options)
   );
});

// Click Event for Push Notifications
self.addEventListener("notificationclick", (event) => {
   event.notification.close();
   event.waitUntil(
      clients.openWindow("http://localhost:5500/")
   );
});
```

Fetch Code :

```
self.addEventListener("fetch", (event) => {
  console.log("[Service Worker] Fetching:", event.request.url);

  if (event.request.method !== "GET") return; // Skip non-GET requests

  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request).then((networkResponse) => {
        return networkResponse;
      });
    }).catch(() => console.warn("[Service Worker] Network request failed:",
event.request.url))
  );
});

// Sync Event - Send Stored Data When Online
self.addEventListener("sync", (event) => {
  if (event.tag === SYNC_TAG) {
    console.log("[Service Worker] Sync event triggered:", SYNC_TAG);
    event.waitUntil(syncOfflineData());
  }
});

// Function to Sync Offline Data & Save to JSON
async function syncOfflineData() {
  const storedData = await getFromLocalStorage();
  if (!storedData.length) {
    console.log("[Service Worker] No offline data to sync.");
    return;
  }

  console.log("[Service Worker] Attempting to sync offline data...", storedData);

  try {
    const response = await fetch("http://localhost:3000/saveData", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(storedData)
    });

    if (response.ok) {
      console.log("[Service Worker] Successfully synced data:", storedData);
      await saveDataToJson(storedData); // Save data to JSON file
      clearLocalStorage(); // Clear after successful sync
    } else {
      console.warn("[Service Worker] Sync failed. Server response not OK.");
    }
  } catch (error) {
    console.error("[Service Worker] Sync failed:", error);
  }
}
// Function to Retrieve Data from Local Storage
function getFromLocalStorage() {
    return new Promise((resolve) => {
        const data = localStorage.getItem("offlineData");
        resolve(data ? JSON.parse(data) : []);
    });
}

// Function to Clear Local Storage after Sync
function clearLocalStorage() {
    localStorage.removeItem("offlineData");
    console.log("[Service Worker] Local storage cleared after sync.");
}

// Save Synced Data to a JSON File
async function saveDataToJson(data) {
    try {
        const response = await fetch(JSON_FILE_URL, {
            method: "PUT", // Use PUT or POST based on server support
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify(data)
        });

        if (response.ok) {
            console.log("[Service Worker] Data saved to JSON file:", JSON_FILE_URL);
        } else {
            console.error("[Service Worker] Failed to save data to JSON.");
        }
    } catch (error) {
        console.error("[Service Worker] Error saving to JSON:", error);
    }
}
```

Output:



Service workers

✓ Offline ✓ Update on reload ☐ Bypass for network

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

Source service-worker.js 2

Received 3/25/2025, 10:06:56 PM

Status ● #2983 activated and is running Stop

Push `Test push message from DevTools.` Push

Sync `test-tag-from-devtools` Sync

Periodic sync `test-tag-from-devtools` Periodic sync

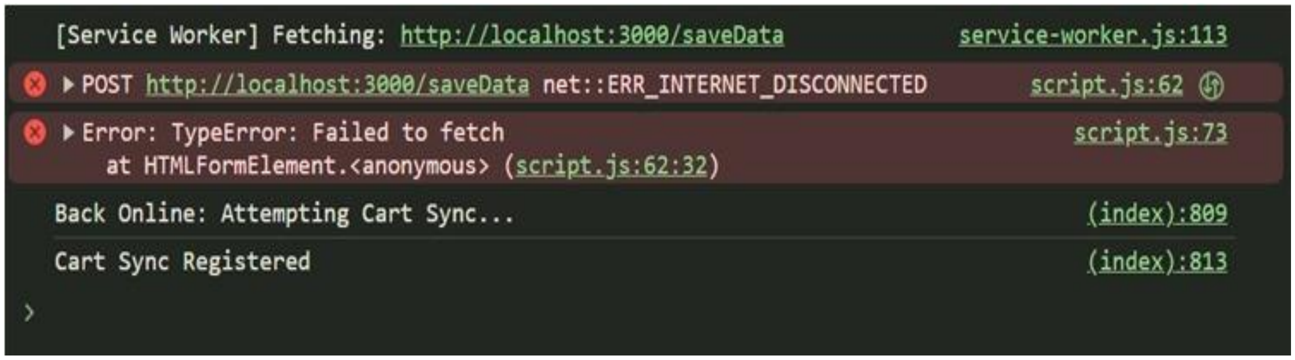
Update Cycle

Version	Update Activity	Timeline
#2983	Install	
#2983	Wait	
#2983	Activate	

Service workers from other origins

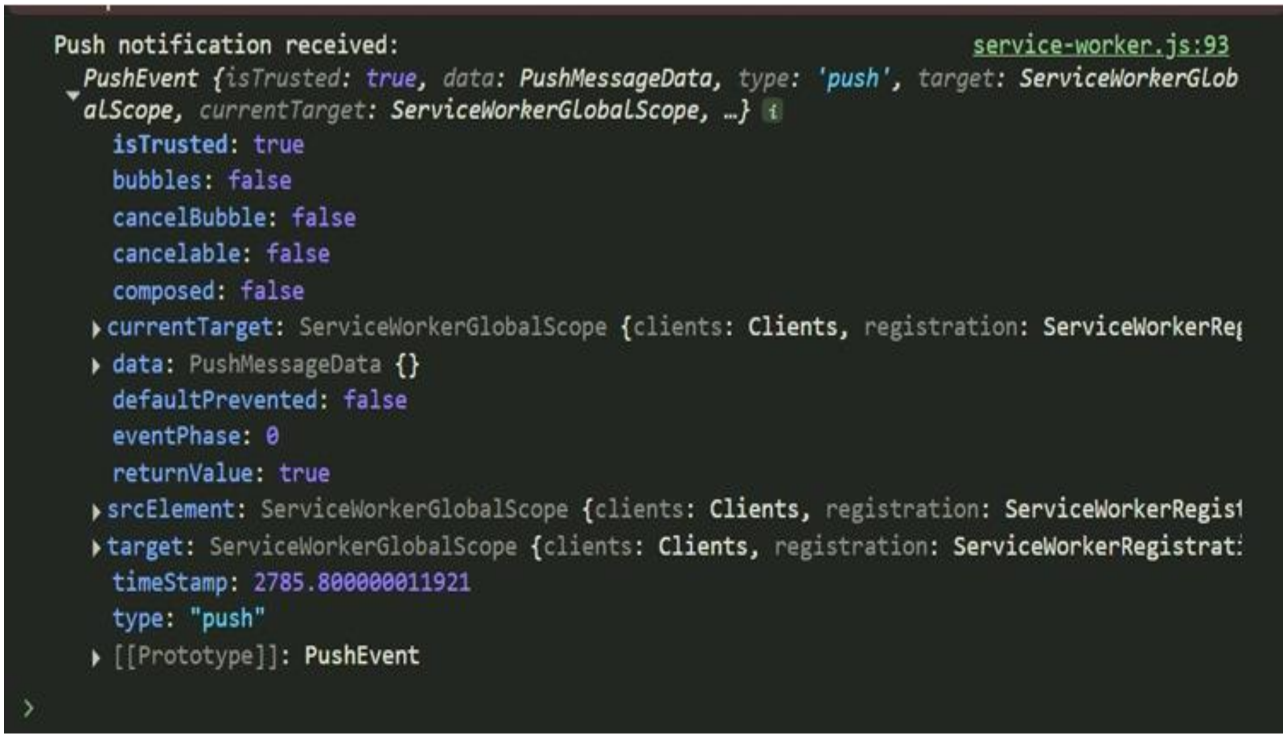
See all registrations

Sync Event

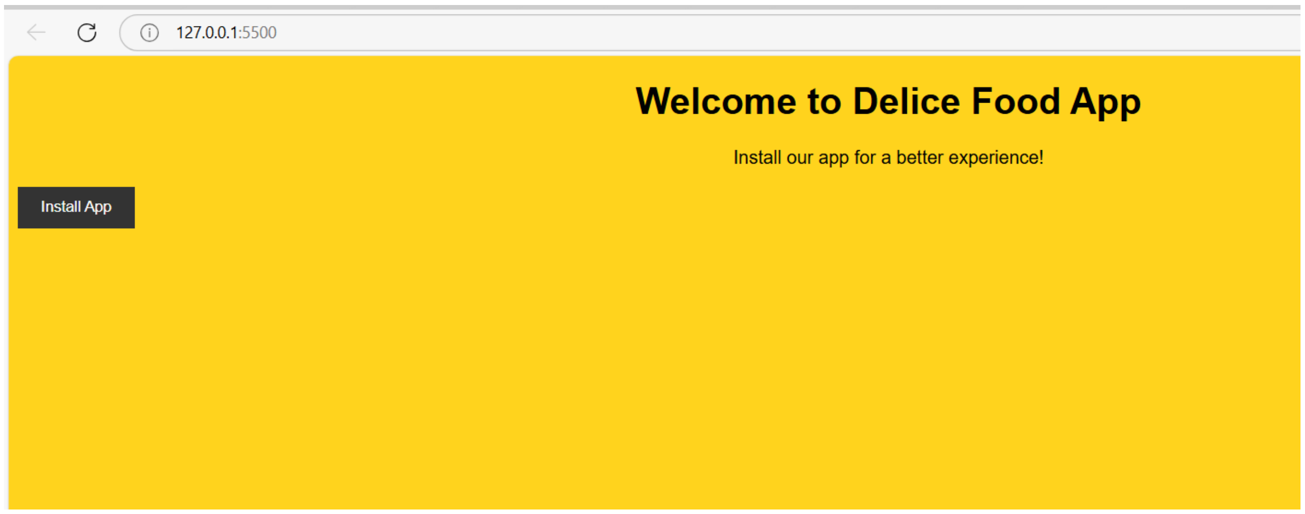


```
[Service Worker] Fetching: http://localhost:3000/saveData          service-worker.js:113
▶ POST http://localhost:3000/saveData net::ERR_INTERNET_DISCONNECTED        script.js:62
▶ Error: TypeError: Failed to fetch                                          script.js:73
    at HTMLFormElement.<anonymous> (script.js:62:32)
Back Online: Attempting Cart Sync...                                       (index):809
Cart Sync Registered                                                       (index):813
>
```

Push event



```
Push notification received:                                          service-worker.js:93
 ▼PushEvent {isTrusted: true, data: PushMessageData, type: 'push', target: ServiceWorkerGlob
  alScope, currentTarget: ServiceWorkerGlobalScope, …}
     isTrusted: true
     bubbles: false
     cancelBubble: false
     cancelable: false
     composed: false
   ▶currentTarget: ServiceWorkerGlobalScope {clients: Clients, registration: ServiceWorkerRe
   ▶ data: PushMessageData {}
     defaultPrevented: false
     eventPhase: 0
     returnValue: true
   ▶srcElement: ServiceWorkerGlobalScope {clients: Clients, registration: ServiceWorkerRegis
   ▶target: ServiceWorkerGlobalScope {clients: Clients, registration: ServiceWorkerRegistrat
     timeStamp: 2785.800000011921
     type: "push"
   ▶ [[Prototype]]: PushEvent
>
```



Welcome to Delice Food App

Install our app for a better experience!