DATE:

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.

Things to note about service worker-
- 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.
- Service workers make extensive use of promises, so if you are new to promises, then you stop reading this and check out promises.

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.

DATE:

- Sync event -
Background Sync is web API that is used to delay a process until the internet connection is stable.

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

FOR EDUCATIONAL USE

Implementation :

Service-worker.js / sw.js

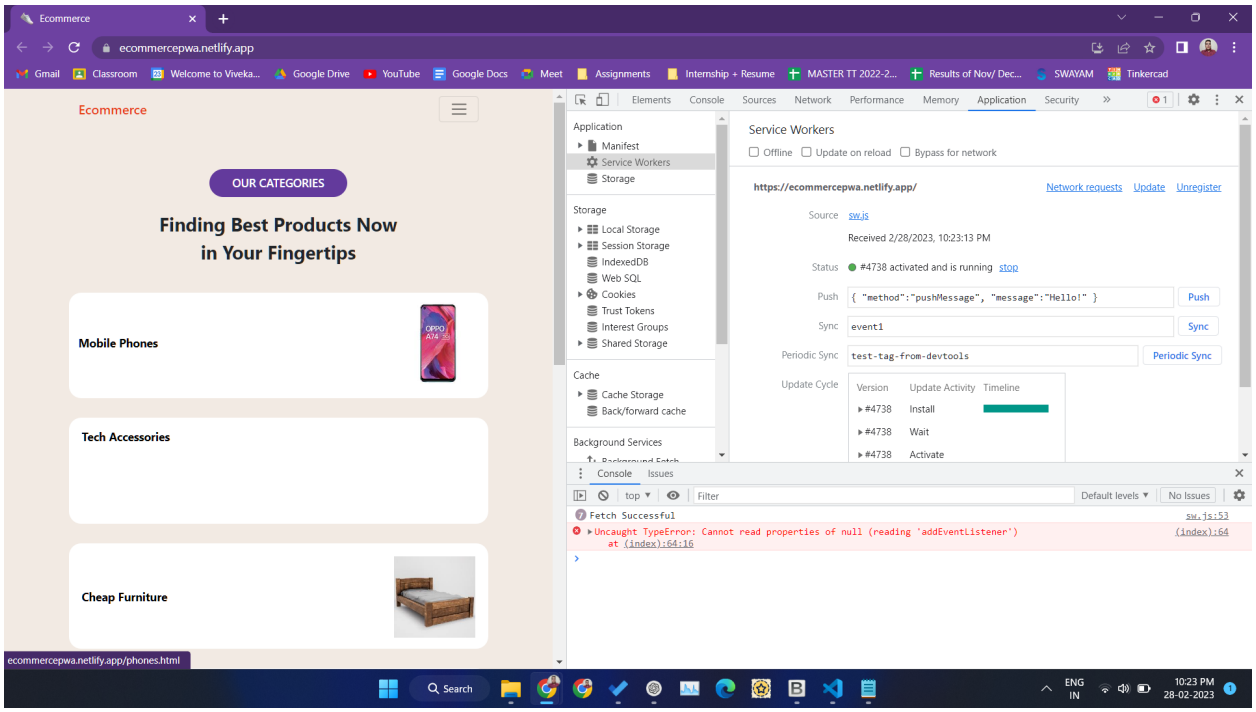
```
const cacheName = "Ecommerce";
const staticAssets = [
  "./",
  "./index.html",
  "./about.html",
  "./drip.html",
  "./electronics.html",
  "./furniture.html",
  "./general.html",
  "./index.html",
  "./laptops.html",
  "./phones.html",
  "./sneakers.html",
  "./manifest.json",
  "./style.css",
];

// Cache static assets on install
self.addEventListener("install", async () => {
  const cache = await caches.open(cacheName);
  await cache.addAll(staticAssets);
  return self.skipWaiting();
});

// Activate service worker and claim clients
self.addEventListener("activate", () => {
  self.clients.claim();
});

// Serve assets from cache first, then from network
self.addEventListener("fetch", async (event) => {
```

```
  const request = event.request;
  const url = new URL(request.url);

  if (url.origin === location.origin) {
    event.respondWith(cacheFirst(request));
  } else {
    event.respondWith(networkAndCache(request));
  }
});

async function cacheFirst(request) {
  const cache = await caches.open(cacheName);
  const cached = await cache.match(request);
  return cached || fetch(request);
}

async function networkAndCache(request) {
  const cache = await caches.open(cacheName);
  try {
    const response = await fetch(request);
    await cache.put(request, response.clone());
    console.log("Fetch Successful");
    return response;
  } catch (error) {
    const cached = await cache.match(request);
    return cached;
  }
}

// Handle push notifications
self.addEventListener("push", function (event) {
  if (event && event.data) {
    const data = event.data.json();
    if (data.method === "pushMessage") {
      console.log("Push notification sent");
```

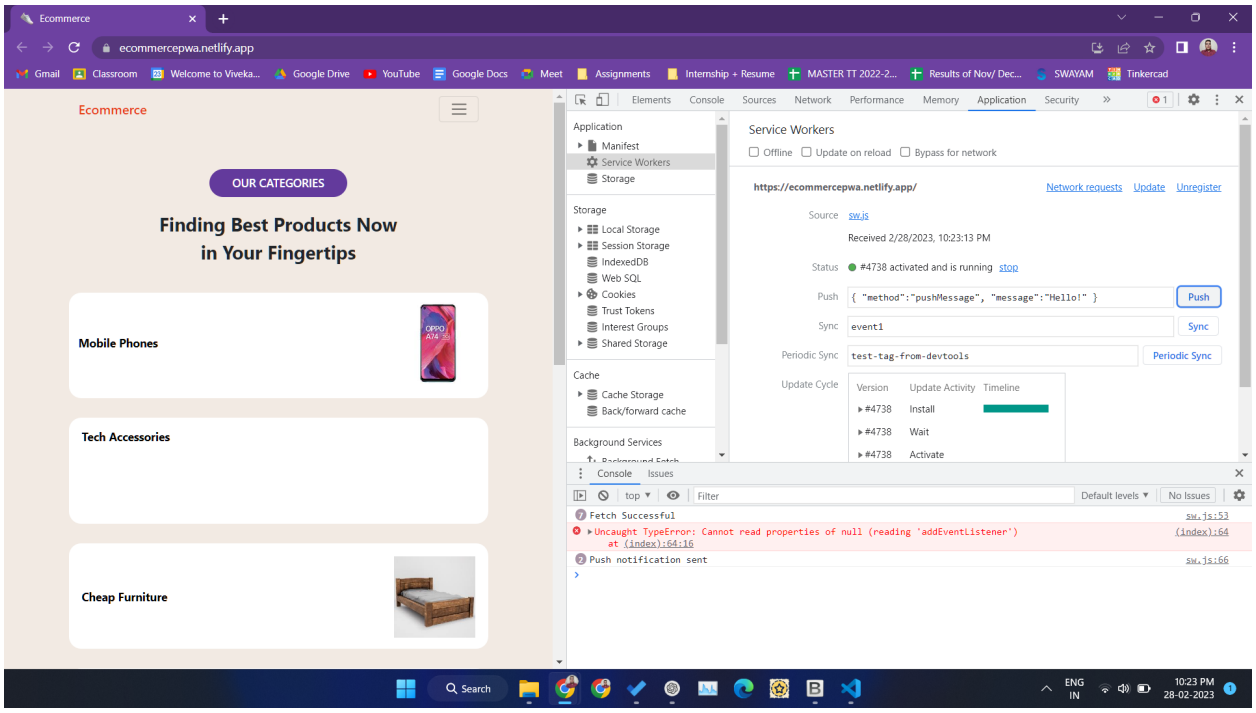
```
      event.waitUntil(
        self.registration.showNotification(" ", {
          body: data.message,
          icon: "path/to/icon.png",
        })
      );
    }
  }
});

// Handle background sync
self.addEventListener("sync", (event) => {
  if (event && event.tag === "event1") {
    console.log("Sync successful!");
    event.waitUntil(
      self.registration.showNotification(" ", {
        body: "Message sent successfully!",
        icon: "path/to/icon.png",
      })
    );
  }
});
```

Output :

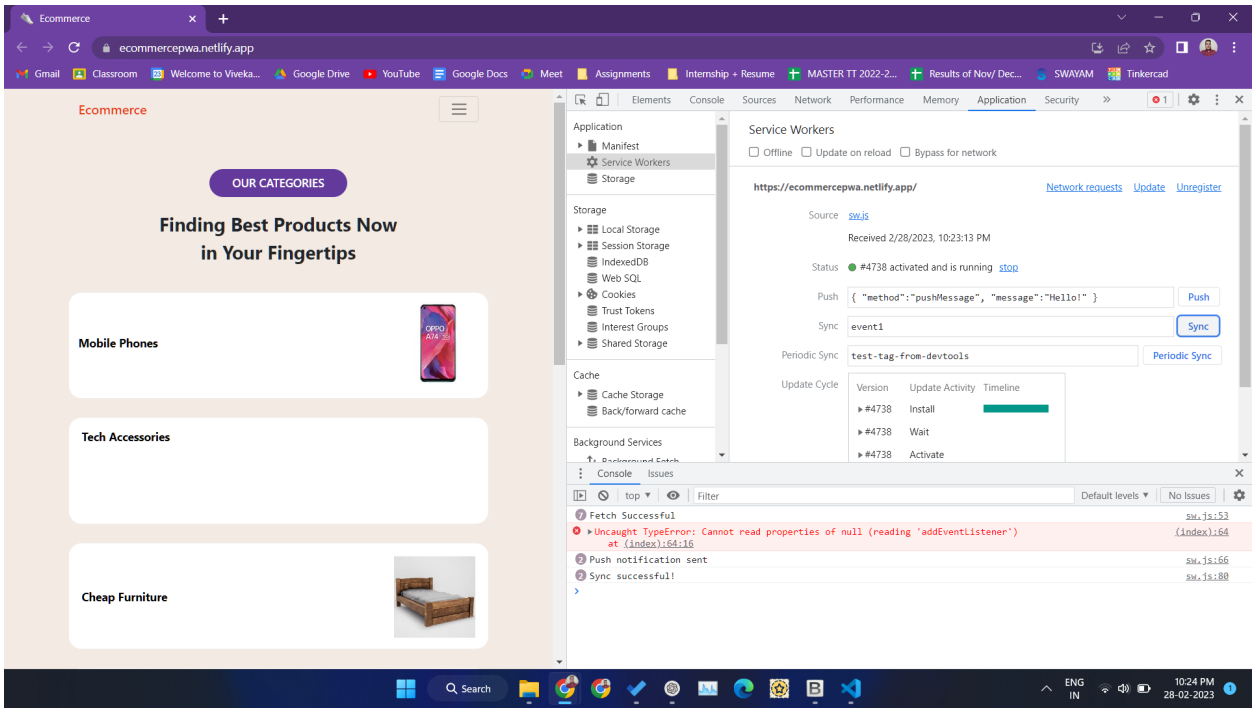
Fetch Successful



Push event



Sync event



Conclusion - We have understood how to implement service worker events such as fetch, push and sync.