Name :Yajat Bhasin **Roll No:5** **Div:D15B**

EXPERIMENT 9

MAD & PWA LAB

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.

Implementation :

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",
];
self.addEventListener("install", async () => {
  const cache = await caches.open(cacheName);
  await cache.addAll(staticAssets);
  return self.skipWaiting();
});
self.addEventListener("activate", () => {
  self.clients.claim();
});
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",
        })
      );
    }
  }
});
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",
```

```
      })
    );
  }
});
```

1.Fetch Event:

The fetch event is triggered whenever a network request is made by the web application. This event allows you to intercept these requests, enabling you to customize how resources are fetched and served. In the context of PWAs, the fetch event is commonly used for:

- Caching Resources: You can intercept requests for static assets such as HTML, CSS, JavaScript files, images, and API responses. By caching these resources, you can improve the performance and offline capabilities of your PWA. This is typically done using service workers

- Offline Support: By caching resources with the fetch event, PWAs can continue to function even when the device is offline or experiencing poor network connectivity. Users can access cached content, providing a seamless experience regardless of their network status.

2.Push Event: The push event allows PWAs to receive and handle push notifications sent from a server. Push notifications are a powerful tool for engaging users and keeping them informed about updates, promotions, or relevant content.

Key aspects of the push event in PWAs include:

● Real-time Updates: Push notifications enable PWAs to deliver real-time updates to users, even when the app is not actively being used. This can include notifications about new messages, order status updates, upcoming events, or personalized recommendations.

● Re-engagement: Push notifications can re-engage users by bringing them back to the PWA with timely and relevant notifications. This helps maintain user interest and encourages repeat visits to the app.

● Personalization: Push notifications can be personalized based on user preferences, behavior, or location, allowing PWAs to deliver targeted and relevant content to individual users.

Name :Yajat Bhasin **Roll No:5** **Div:D15B**

3. Sync Event: The sync event allows PWAs to perform background synchronization tasks. This event is particularly useful for scenarios where data needs to be synchronized with a server, but the device may not be online at the time of the update.

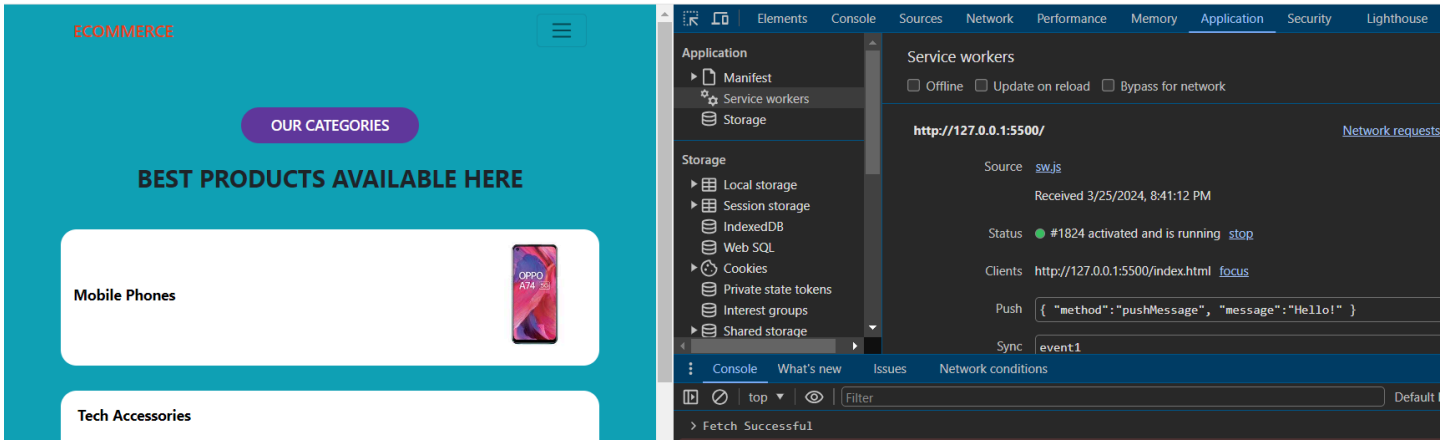
Key use cases for the sync event in PWAs include:

● Background Data Sync: PWAs can use the sync event to periodically synchronize data with a server in the background. This is useful for updating user data, such as syncing changes to a shopping cart, updating user preferences, or sending analytics data.

● Offline Task Queuing: If a user performs actions while offline (e.g., adding items to a shopping cart), those actions can be queued using the sync event and executed once the device regains connectivity. This ensures that no data is lost even if the user goes offline temporarily.

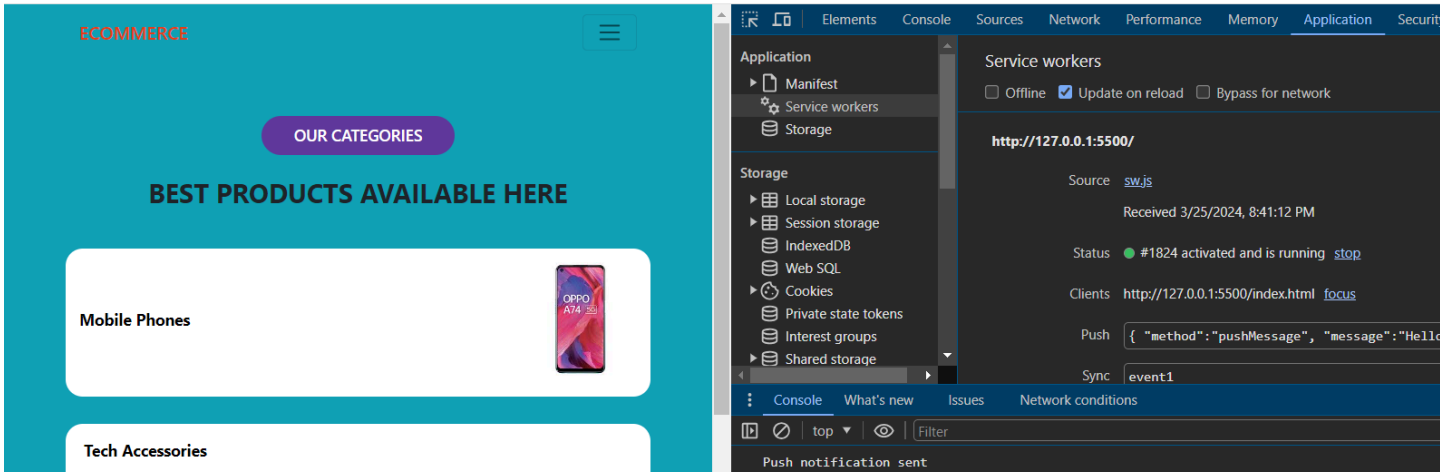
Output :

Fetch Successful

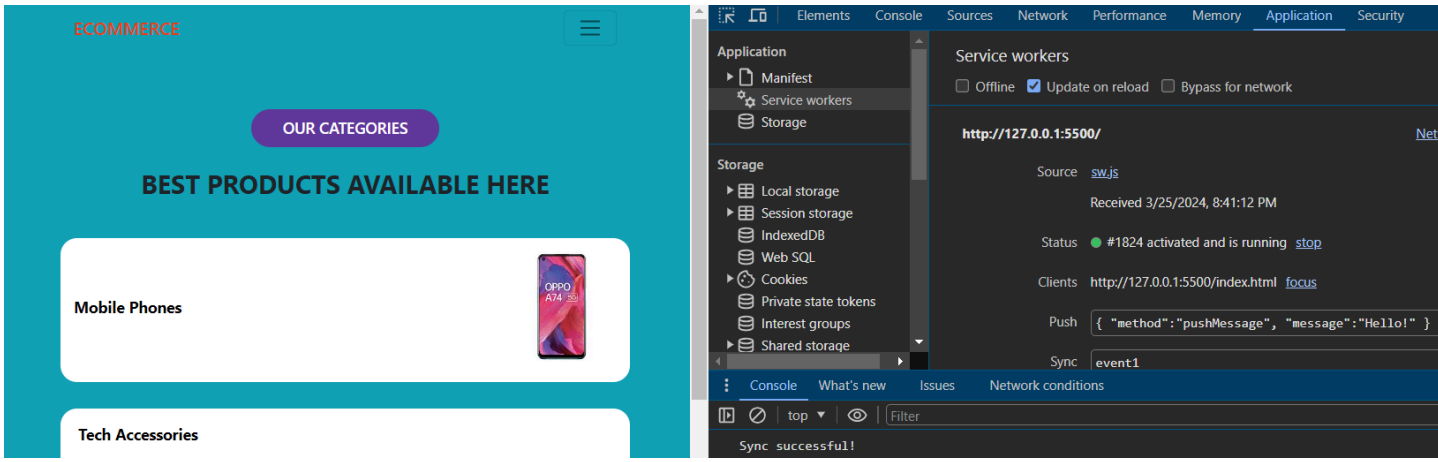


Name :Yajat Bhasin **Roll No:5** **Div:D15B**

Push event



Sync event



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