Name : Nilanchala Panda **Batch : B**
Div . : 41 **Roll No . : 41**

EXPERIMENT NO 9

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.

CODE -

index.html -

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>E-commerce PWA</title>
    <link rel="stylesheet" href="main.css" />
  </head>
  <body>
    <h1>Welcome to Our E-commerce Store</h1>
    <!-- E-commerce content goes here -->
    <script src="app.js"></script>
  </body>
</html>
```

service-worker.js -

```
const CACHE_NAME = "my-ecommerce-app-cache-v1";
const urlsToCache = [
  "/",
  "index.html",
  "service-worker.js",
  "manifest.json",
];

var staticCacheName = "pwa";
```

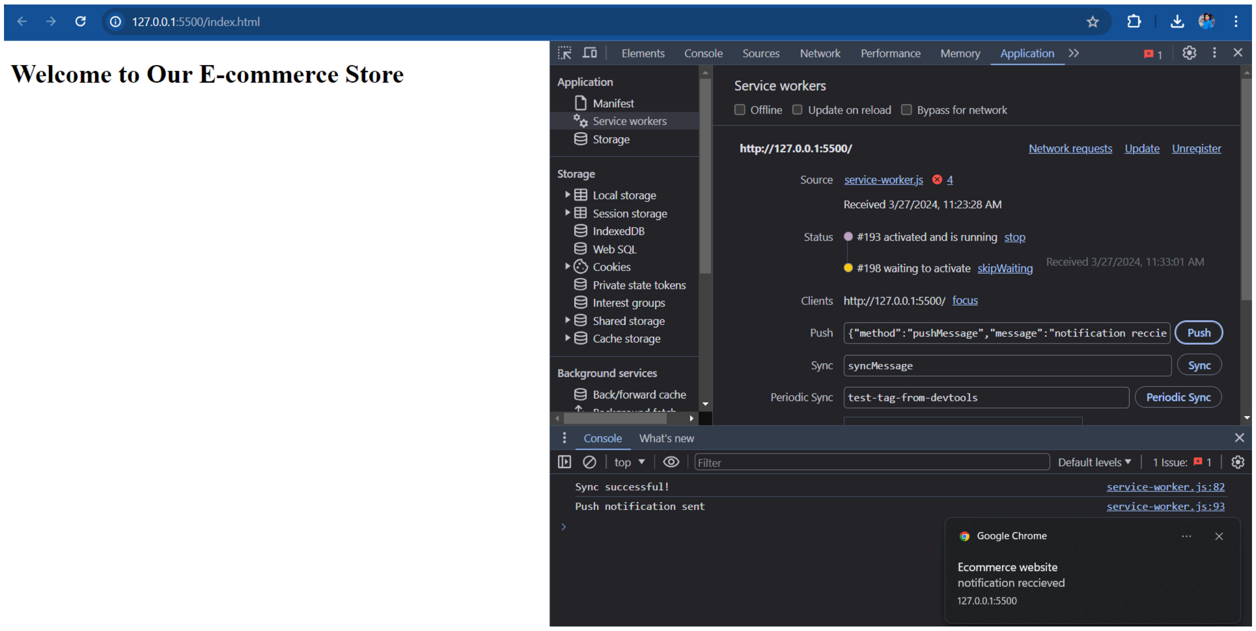
```
self.addEventListener("install", function (e) {
  e.waitUntil(
    caches.open(staticCacheName).then(function (cache) {
      return cache.addAll(["/"]);
    })
  );
});

self.addEventListener("fetch", function (event) {
  console.log(event.request.url);

  event.respondWith(
    caches.match(event.request).then(function (response) {
      return response || fetch(event.request);
    })
  );
});

self.addEventListener("install", function (event) {
  event.waitUntil(
    caches.open(CACHE_NAME).then(function (cache) {
      console.log("Opened cache");
      return cache.addAll(urlsToCache).catch(function (error) {
        console.error("Cache.addAll error:", error);
      });
    })
  );
});

self.addEventListener("activate", function (event) {
  // Perform activation steps
  event.waitUntil(
    caches.keys().then(function (cacheNames) {
      return Promise.all(
        cacheNames.map(function (cacheName) {
          if (cacheName !== CACHE_NAME) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});

// Fetch event listener
self.addEventListener("fetch", function (event) {
  event.respondWith(
    checkResponse(event.request).catch(function () {
      console.log("Fetch from cache successful!");
      return returnFromCache(event.request);
    })
  );
  console.log("Fetch successful!");
  event.waitUntil(addToCache(event.request));
});

// Sync event listener
self.addEventListener("sync", function (event) {
  if (event.tag === "syncMessage") {
    console.log("Sync successful!");
  }
});

// Push event listener
self.addEventListener("push", function (event) {
  if (event && event.data) {
    try {
      var data = event.data.json();
      if (data && data.method === "pushMessage") {
        console.log("Push notification sent");
        self.registration.showNotification("Ecommerce website", {
          body: data.message,
        });
      }
    } catch (error) {
      console.error("Error parsing push data:", error);
    }
  }
});
```

```
self.addEventListener("activate", async () => {
  if (Notification.permission !== "granted") {
    try {
      const permission = await Notification.requestPermission();
      if (permission === "granted") {
        console.log("Notification permission granted.");
      } else {
        console.warn("Notification permission denied.");
      }
    } catch (error) {
      console.error("Failed to request notification permission:", error);
    }
  }
});

var checkResponse = function (request) {
  return new Promise(function (fulfill, reject) {
    fetch(request)
      .then(function (response) {
        if (response.status !== 404) {
          fulfill(response);
        } else {
          reject(new Error("Response not found"));
        }
      })
      .catch(function (error) {
        reject(error);
      });
  });
};

var returnFromCache = function (request) {
  return caches.open("offline").then(function (cache) {
    return cache.match(request).then(function (matching) {
      if (!matching || matching.status == 404) {
        return cache.match("offline.html");
      } else {
        return matching;
      }
    });
  });
};

var addToCache = function (request) {
  return caches.open("offline").then(function (cache) {
    return fetch(request).then(function (response) {
      return cache.put(request, response.clone()).then(function () {
        return response;
      });
    });
  });
};
```

app.js -

```
if ("Notification" in window) {
  Notification.requestPermission().then(function (permission) {
    if (permission === "granted") {
      console.log("Notification permission granted.");
    } else {
      console.warn("Notification permission denied.");
    }
  });
}
```

OUTPUT -



Welcome to Our E-commerce Store

Service workers

Offline ☐ Update on reload ☐ Bypass for network ☐

http://127.0.0.1:5500/

Source: service-worker.js 4

Received 3/27/2024, 11:23:28 AM

Status: #193 activated and is running stop

#198 waiting to activate skipWaiting Received 3/27/2024, 11:33:01 AM

Clients: http://127.0.0.1:5500/ focus

Push: {"method":"pushMessage","message":"notification reccie

Sync: syncMessage

Periodic Sync: test-tag-from-devtools

```
Sync successful!                                         service-worker.js:82
Push notification sent                                   service-worker.js:93
```

Google Chrome

Ecommerce website
notification recciеved
127.0.0.1:5500