

Experiment No. 9

Aim: To implement Service Worker events like fetch, sync, and push for an E-commerce Progressive Web Application (PWA).

Theory:

Service Worker:

A Service Worker is a script operating in the background of a browser without user interaction, akin to a proxy on the user's side. It facilitates tracking network traffic, managing push notifications, and developing "offline-first" web applications using Cache API.

Key Points about Service Worker:

- It acts as a programmable network proxy to control how page network requests are handled.
- Service workers function exclusively over HTTPS to mitigate potential "man-in-the-middle" attacks.
- They become idle when not in use and restart when needed. Global states do not persist between events, but IndexedDB databases can be utilized for persistent data.
- Extensive use of promises is made, making familiarity with promises crucial for effective implementation.

Fetch Event:

This event enables tracking and management of page network traffic. It offers flexibility in managing "cache first" and "network first" requests.

- CacheFirst: If the requested content is cached, the cached response is returned; otherwise, a new response is fetched from the network.
- NetworkFirst: Attempts to retrieve an updated response from the network; if unsuccessful, checks for cached responses. If no cache exists, customizable actions can be taken, such as returning dummy content or informational messages.

Sync Event:

Background Sync, a Web API, defers processes until a stable internet connection is available. This concept can be likened to sending an email in an email client application when the internet connection is unreliable.

Push Event:

Handles push notifications received from the server. The example demonstrates a straightforward approach of displaying a notification when a "pushMessage" method is received.

Code: sw.js

```
var filesToCache = [ '/pages/index.html', '/styles/style.css', '/images/Work.png', '/manifest.json',
'/offline.html', // Add the offline page to be cached
// Add other files and routes you want to cache here
];

// Function to cache files during installation var preLoad = function() {
return caches.open("ecommerce-app").then(function(cache) { return cache.addAll(filesToCache);
});
};

// Service Worker installation event self.addEventListener("install", function(event) {
event.waitUntil(preLoad());
});

// Service Worker fetch event// Service Worker fetch event self.addEventListener("fetch",
function(event) {
event.respondWith(
// Check if request is in cache caches.match(event.request).then(function(response) {
// If request is found in cache, return it if (response) {
return response;
}
// If request is not found in cache, fetch it from the network return
fetch(event.request).then(function(response) {
// Check if the request URL scheme is supported for caching if
(!event.request.url.startsWith('http')) {
return response;
}
// Clone the response to cache it
var responseToCache = response.clone();
// Cache the fetched response for future use
caches.open("ecommerce-app").then(function(cache) {
cache.put(event.request, responseToCache);
});
return response;
}).catch(function() {
// If fetching from network fails, return the offline page return caches.match("/offline.html");
});
});

// Service Worker sync event self.addEventListener('sync', function(event) {
if (event.tag === 'syncData') { event.waitUntil(syncData());
```

```
}  
});  
// Function to synchronize data in the background function syncData() {  
// Implement data synchronization logic here  
// For example, send pending orders to the server  
}  
  
// Service Worker push event  
// Service Worker push event self.addEventListener('push', function(event) {  
if (event && event.data) {  
var data = event.data.json();  
if (data.method === "pushMessage") { console.log("Push notification sent");  
// Request permission to show notifications self.registration.showNotification("Abhishek's  
MBAKARO", {  
body: data.message  
});  
  
}  
});  
...
```

Output:

- Fetch event
- Sync event
- Push event

o/p:

The screenshot displays the Chrome DevTools Application tab for a service worker. The left sidebar shows the 'Application' panel with 'Service workers' selected. The main area shows the configuration for the service worker at `http://127.0.0.1:5500/pages/`. The status is '#2696 activated and is running'. The console shows two log messages: 'ServiceWorker registration successful with scope: http://127.0.0.1:5500/pages/' and 'Push notification sent'.

Service workers

☐ Offline ☐ Update on reload ☐ Bypass for network

http://127.0.0.1:5500/pages/ [Network requests](#) [Update](#) [Unregister](#)

Source [service-worker.js](#) 1

Received 27/03/2024, 20:45:32

Status ● #2696 activated and is running [stop](#)

Push [Push](#)

Sync [Sync](#)

Periodic Sync [Periodic Sync](#)

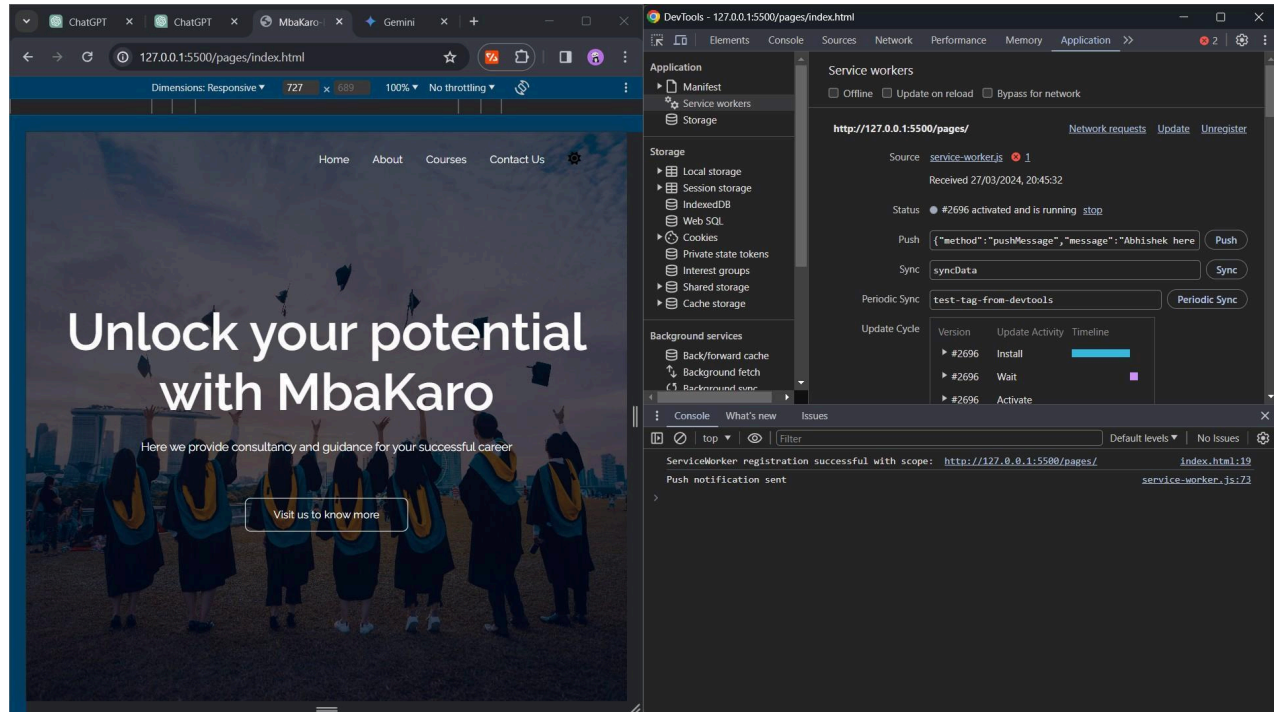
Update Cycle

Version	Update Activity	Timeline
▶ #2696	Install	<div></div>
▶ #2696	Wait	<div></div>
▶ #2696	Activate	<div></div>

Console What's new Issues

ServiceWorker registration successful with scope: <http://127.0.0.1:5500/pages/> [index.html:19](#)

Push notification sent [service-worker.js:73](#)

**Conclusion:**

We have effectively implemented and comprehended Service Worker events such as fetch, push, and sync for our e-commerce Progressive Web App (PWA).