

## Experiment 9

**Aim:** To code and register a service worker, and complete the install and activation process for a new service worker for the 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.

Here is a job for the Background Sync.

The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.

Here, you can create any scenario for yourself. A sample is in the following for this case.

When we click the "send" button, email content will be saved to IndexedDB.

Background Sync registration.

If the Internet connection is available, all email content will be read and sent to Mail Server.

If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

You can see the working process within the following code block.

Event Listener for Background Sync Registration

### **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.

You can use Application Tab from Chrome Developer Tools for testing push notification.

Code:

**Event Listener for Background Sync Registration:**

```
// Listen for the 'sync' event
self.addEventListener("sync", function (event) {
  if (event.tag === "background-sync-tag") {
    // Perform background sync tasks here
    console.log("Background sync event triggered.");
    event.waitUntil(doBackgroundSync());
  }
});

// Function to perform background sync tasks
function doBackgroundSync() {
  // Implement your background sync logic here
  console.log("Performing background sync...");
}
```

### Push Event

The Push Event is triggered when a push notification is received from the server. It allows developers to handle the incoming data and perform actions based on the received notification. For example, in a service worker (sw.js), you can use the Push Event to display a notification to the user. First, you would request permission to show notifications using `Notification.requestPermission()`. Then, when a push notification is received, you can extract the message from the notification and display it to the user using the `showNotification()` method.

```
// Listen for the 'push' event
self.addEventListener('push', function(event) {
  console.log('Push event received.');
```

```
  // Parse the notification data
  let notificationData = {};
  if (event.data) {
    notificationData = event.data.json();
  }

  // Show the notification
  event.waitUntil(
    self.registration.showNotification(notificationData.title, {
      body: notificationData.body,
      icon: notificationData.icon,
      // You can add more options here as needed
    })
  );
});
```

```
// Service worker code

// Listen for the 'install' event
self.addEventListener("install", function (event) {
  console.log("Service Worker installed");
});

// Listen for the 'activate' event
self.addEventListener("activate", function (event) {
  console.log("Service Worker activated");
});

// Listen for the 'fetch' event
self.addEventListener("fetch", function (event) {
  console.log("Fetch event intercepted:", event.request.url);

  // Respond to the fetch request
  event.respondWith(
    // Try fetching the resource from the network
    fetch(event.request)
      .then(function (response) {
        // If fetch is successful, clone the response
        if (response && response.status === 200 && response.type === "basic") {
          // Clone the response before returning it
          let responseToCache = response.clone();

          // You can perform caching or other processing here if needed

          // Return the original response
          return response;
        } else {
          // If fetch fails, return a fallback response
          return new Response("Fallback response", { status: 404 });
        }
      })
      .catch(function (error) {
        // If fetch fails, return a fallback response
        console.error("Fetch failed:", error);
        return new Response("Fallback response", { status: 404 });
      })
  );
});

// Listen for the 'sync' event
```

```
self.addEventListener("sync", function (event) {
  if (event.tag === "background-sync-tag") {
    // Perform background sync tasks here
    console.log("Background sync event triggered.");
    event.waitUntil(doBackgroundSync());
  }
});

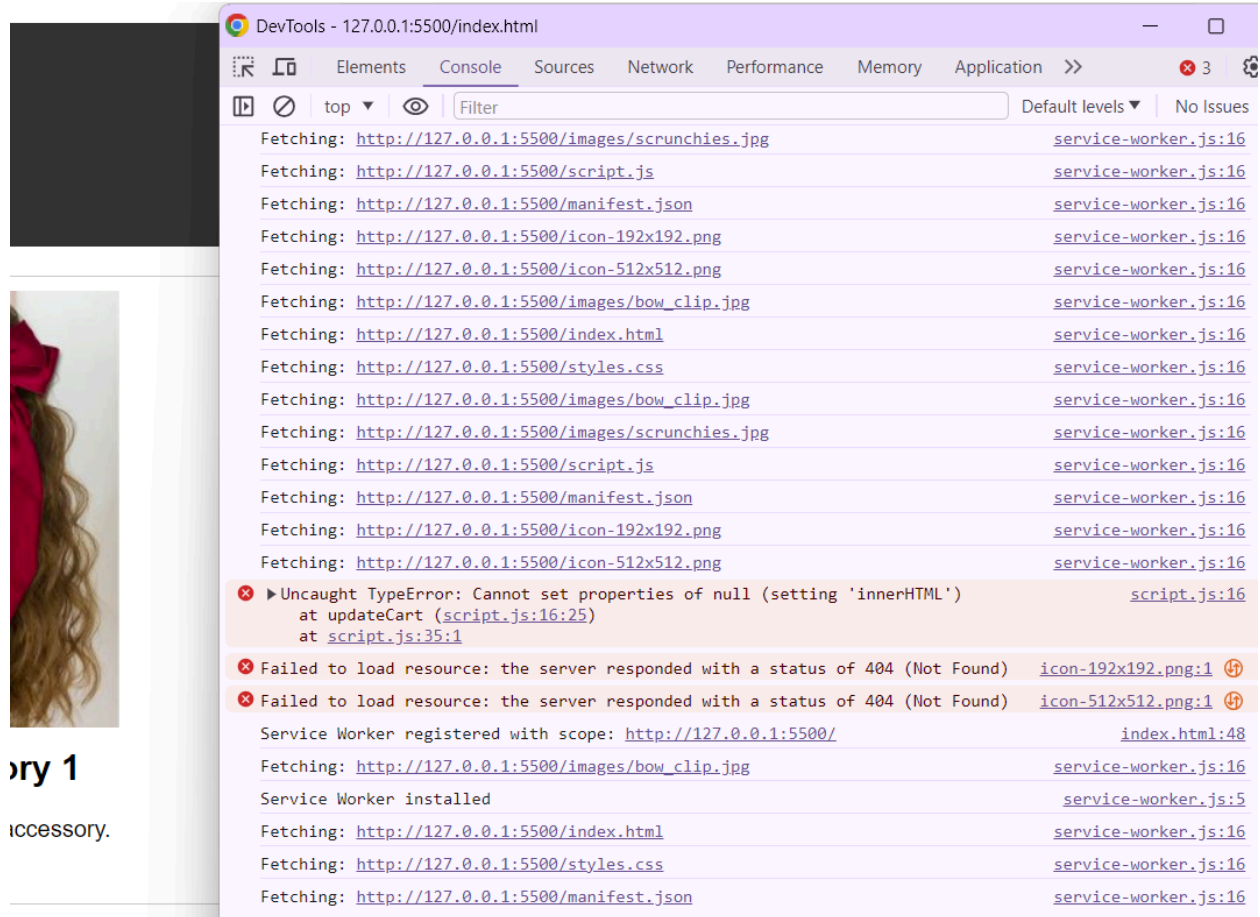
// Function to perform background sync tasks
function doBackgroundSync() {
  // Implement your background sync logic here
  console.log("Performing background sync...");
}

// Listen for the 'push' event
self.addEventListener("push", function (event) {
  console.log("Push event received.");

  // Parse the notification data
  let notificationData = {};
  if (event.data) {
    notificationData = event.data.json();
  }

  // Show the notification
  event.waitUntil(
    self.registration.showNotification(notificationData.title, {
      body: notificationData.body,
      icon: notificationData.icon,
      // You can add more options here as needed
    })
  );
});
```

**Output:****FetchEvent**



DevTools - 127.0.0.1:5500/index.html

Elements Console Sources Network Performance Memory Application >> 3 No Issues

Filter

Fetching: <http://127.0.0.1:5500/images/scrunchies.jpg> service-worker.js:16

Fetching: <http://127.0.0.1:5500/script.js> service-worker.js:16

Fetching: <http://127.0.0.1:5500/manifest.json> service-worker.js:16

Fetching: <http://127.0.0.1:5500/icon-192x192.png> service-worker.js:16

Fetching: <http://127.0.0.1:5500/icon-512x512.png> service-worker.js:16

Fetching: [http://127.0.0.1:5500/images/bow\\_clip.jpg](http://127.0.0.1:5500/images/bow_clip.jpg) service-worker.js:16

Fetching: <http://127.0.0.1:5500/index.html> service-worker.js:16

Fetching: <http://127.0.0.1:5500/styles.css> service-worker.js:16

Fetching: [http://127.0.0.1:5500/images/bow\\_clip.jpg](http://127.0.0.1:5500/images/bow_clip.jpg) service-worker.js:16

Fetching: <http://127.0.0.1:5500/images/scrunchies.jpg> service-worker.js:16

Fetching: <http://127.0.0.1:5500/script.js> service-worker.js:16

Fetching: <http://127.0.0.1:5500/manifest.json> service-worker.js:16

Fetching: <http://127.0.0.1:5500/icon-192x192.png> service-worker.js:16

Fetching: <http://127.0.0.1:5500/icon-512x512.png> service-worker.js:16

✖ ▶ Uncaught TypeError: Cannot set properties of null (setting 'innerHTML')  
at updateCart (script.js:16:25)  
at script.js:35:1 service-worker.js:16

✖ Failed to load resource: the server responded with a status of 404 (Not Found) icon-192x192.png:1

✖ Failed to load resource: the server responded with a status of 404 (Not Found) icon-512x512.png:1

Service Worker registered with scope: <http://127.0.0.1:5500/> index.html:48

Fetching: [http://127.0.0.1:5500/images/bow\\_clip.jpg](http://127.0.0.1:5500/images/bow_clip.jpg) service-worker.js:16

Service Worker installed service-worker.js:5

Fetching: <http://127.0.0.1:5500/index.html> service-worker.js:16

Fetching: <http://127.0.0.1:5500/styles.css> service-worker.js:16

Fetching: <http://127.0.0.1:5500/manifest.json> service-worker.js:16

The screenshot shows the Chrome DevTools Application tab with the 'Service workers' section selected. The left sidebar lists various application components: Manifest, Service workers (selected), Storage, and Background services. The main panel displays details for a service worker registered at <http://127.0.0.1:5500/>. It shows the source as [service-worker.js](#) with 11 errors. The status indicates that version #3380 is activated and running, while version #3385 is waiting to activate. The 'Update Cycle' section provides a detailed timeline of the installation, waiting, and activation phases for version #3380.

**Service workers**

☐ Offline ☒ Update on reload ☐ Bypass for network

**<http://127.0.0.1:5500/>** [Network requests](#) [Update](#) [Unregister](#)

Source [service-worker.js](#) ✖ 11

Received 3/25/2024, 7:38:44 PM

Status ● #3380 activated and is running [stop](#)

● #3385 waiting to activate [skipWaiting](#)

Received 3/25/2024, 7:45:44 PM

Push  [Push](#)

Sync  [Sync](#)

Periodic Sync  [Periodic Sync](#)

Update Cycle

Version	Update Activity	Timeline
▶ #3380	Install	
▶ #3380	Wait	
▶ #3380	Activate	<div></div>

**Service workers from other origins**

[See all registrations](#)

**Conclusion :** I have understood and successfully registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.