

MAD & PWA Lab
Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	20
Name	Madhura Jangale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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

SERVICE WORKER FOR PUSH NOTIFICATIONS :

```
const CACHE_NAME = 'blogbreeze-v1';
const filesToCache = [
  '/',
  'index.html',
  'manifest.json',
  '/src/assets/react.svg',
  '/public/icon.png',
  '/src/index.css',
  '/src/main.jsx',
```

```
];
```

```
// Install event: Caching important files
self.addEventListener('install', (event) => {
  console.log('Service Worker: Installed');
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('Service Worker: Caching files');
        return cache.addAll(filesToCache);
      })
  );
});
```

```
// Activate event: Cleaning up old caches
self.addEventListener('activate', (event) => {
  console.log('Service Worker: Activated');
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME) {
            console.log('Service Worker: Deleting old cache', cacheName);
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

```
// Fetch event: Intercepting requests and serving from cache if available
```

```

self.addEventListener('fetch', (event) => {
  console.log('Service Worker: Fetching', event.request.url);

  event.respondWith(
    caches.match(event.request)
      .then((cachedResponse) => {
        if (cachedResponse) {
          console.log('Service Worker: Returning cached response');
          return cachedResponse; // Return the cached response if available
        }

        return fetch(event.request)
          .then((networkResponse) => {
            caches.open(CACHE_NAME).then((cache) => {
              console.log('Service Worker: Caching new response for', event.request.url);
              cache.put(event.request, networkResponse.clone()); // Cache the new response
            });
            return networkResponse;
          });
      })
  );
});

```

```

self.addEventListener('push', (event) => {
  console.log('Push notification received:', event);

  if (event && event.data) {
    // Parse the push notification data
    const data = event.data.json();

    if (data.method === 'pushMessage') {
      console.log('Push notification sent with message:', data.message);
    }
  }
});

```

```

// Set up notification options (like body, icon, etc.)
const options = {
  body: data.message || 'Hello, this is a default message!', // Default or push message
  icon: '/src/assets/react.svg', // Icon for the notification
  badge: '/src/assets/react.svg', // Badge icon for the notification
};

// Check if the notification permission is granted before showing the notification
if (Notification.permission === 'granted') {
  // Show the notification with a title
  event.waitUntil(
    self.registration.showNotification('Blog Breeze', options)
  );
} else {
  console.log('Notification permission not granted yet.');
}
}
});

```

SERVICE WORKER SCRIPT FOR SYNC AND FETCH :

```

const CACHE_NAME = 'blogbreeze-v1';
const filesToCache = [
  '/',
  'index.html',
  'manifest.json',
  '/src/assets/react.svg',
  '/public/icon.png',
  '/src/index.css',

```

```

'/src/main.jsx',

];

// Install event: Caching important files
self.addEventListener('install', (event) => {
  console.log('Service Worker: Installed');
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => {
        console.log('Service Worker: Caching files');
        return cache.addAll(filesToCache);
      })
  );
});

// Activate event: Cleaning up old caches
self.addEventListener('activate', (event) => {
  console.log('Service Worker: Activated');
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheName !== CACHE_NAME) {
            console.log('Service Worker: Deleting old cache', cacheName);
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});

```

// Fetch event: Intercepting requests and serving from cache if available

```
self.addEventListener('fetch', function (event) {  
  console.log('Service Worker: Fetching', event.request.url);  
  
  // Skip caching for Firebase API requests  
  if (event.request.url.includes('firestore.googleapis.com')) {  
    // Always fetch Firebase data from the network (no cache)  
    event.respondWith(fetch(event.request));  
    return;  
  }  
}
```

```
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: Handling background sync

```
self.addEventListener('sync', function (event) {  
  if (event.tag === 'syncMessage') {  
    console.log('Sync successful!');  
    // You can add more background sync logic here  
  }  
});
```



```
// Placeholder functions for cache and response handling
function checkResponse(request) {
  return caches.match(request)
    .then(function (cachedResponse) {
      if (cachedResponse) {
        return cachedResponse;
      }
      return fetch(request);
    });
}

function returnFromCache(request) {
  return caches.match(request);
}

function addToCache(request) {
  return fetch(request)
    .then(function (response) {
      if (!response || response.status !== 200 || response.type !== 'basic') {
        return response;
      }
      return caches.open(CACHE_NAME)
        .then(function (cache) {
          console.log('Service Worker: Adding to cache', request.url);
          cache.put(request, response);
          return response;
        });
    });
}
```

OUTPUT :

push

The screenshot shows a web browser at <https://eyojana-project-rust.vercel.app/>. The page has a navigation menu with links: Home, Schemes, About, FAQs, Contact Us, and My Applications. A 'Login' button is visible. Below the navigation menu, there is a section titled 'Discovering government schemes that match your needs... According to your eligibility' with a green button labeled 'Login'.

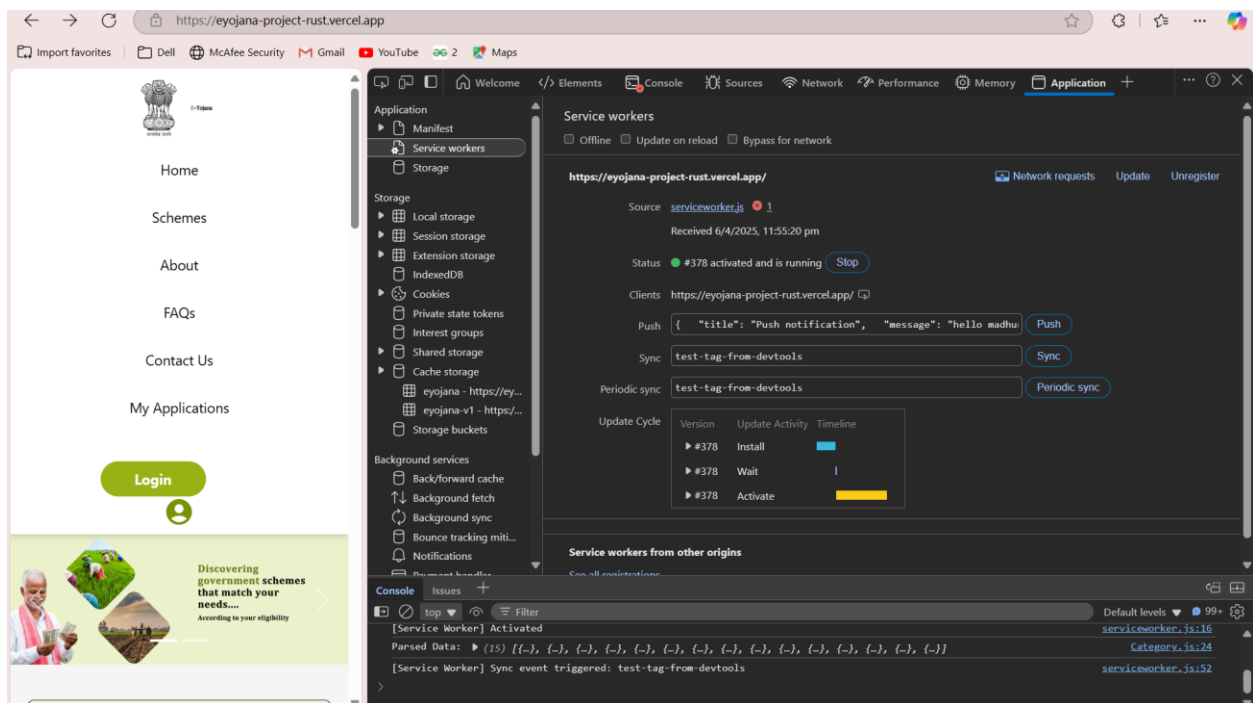
The Chrome DevTools Application panel is open, showing the 'Service workers' section. The service worker is `serviceworker.js` from `https://eyojana-project-rust.vercel.app/`. It is currently active and running. The 'Push' event is triggered, sending a message: `{ "title": "Push notification", "message": "hello madhu" }`. The 'Sync' and 'Periodic sync' events are also shown, both triggered from `test-tag-from-devtools`. The 'Update Cycle' section shows the service worker's lifecycle: `#376 Install`, `#376 Wait`, and `#376 Activate`.

fetch

The screenshot shows the same web browser as above. The Chrome DevTools Network panel is open, displaying a list of network requests. The requests are categorized by type: `Skills.svg`, `Social%20Welfare.svg`, `Sports.svg`, `Transport.svg`, `Travel.svg`, `Utility.svg`, `Women.svg`, `kisang.bc7f29c5ece1ce024000.png`, `2.47649feb1bc72fd37e1c.png`, `Steps.bf949cac48e7bb4d600c.png`, `faq3.93518a0b53a07dcaec14.png`, `kisang.bc7f29c5ece1ce024000.png`, `2.47649feb1bc72fd37e1c.png`, `Steps.bf949cac48e7bb4d600c.png`, `faq3.93518a0b53a07dcaec14.png`, `collect?v=2&tid=G-SLDZGKJS3X&utm=45je5421v9205...`, and `collect?v=2&tid=G-SLDZGKJS3X&utm=45je5421v92...`. The requests are initiated by `serviceworker.js:24` and `react-dom.production.min.js:2`. The 'Fulfilled by' column shows that some requests are fulfilled by the ServiceWorker, while others are fulfilled by the disk cache.

Name	Status	Type	Initiator	Size	Time	Fulfilled by
Skills.svg	200	fetch	serviceworker.js:24	1.9 kB	34 ms	
Social%20Welfare.svg	200	fetch	serviceworker.js:24	2.1 kB	35 ms	
Sports.svg	200	fetch	serviceworker.js:24	2.9 kB	36 ms	
Transport.svg	200	fetch	serviceworker.js:24	1.8 kB	37 ms	
Travel.svg	200	fetch	serviceworker.js:24	3.3 kB	38 ms	
Utility.svg	200	fetch	serviceworker.js:24	2.2 kB	39 ms	
Women.svg	200	fetch	serviceworker.js:24	3.7 kB	39 ms	
kisang.bc7f29c5ece1ce024000.png	200	png	react-dom.production.min.js:2	0 B	36 ms	(ServiceWorker)
2.47649feb1bc72fd37e1c.png	200	png	react-dom.production.min.js:2	0 B	36 ms	(ServiceWorker)
Steps.bf949cac48e7bb4d600c.png	200	png	react-dom.production.min.js:2	0 B	32 ms	(ServiceWorker)
faq3.93518a0b53a07dcaec14.png	200	png	react-dom.production.min.js:2	0 B	34 ms	(ServiceWorker)
kisang.bc7f29c5ece1ce024000.png	200	fetch	serviceworker.js:24	0 B	15 ms	(disk cache)
2.47649feb1bc72fd37e1c.png	200	fetch	serviceworker.js:24	0 B	9 ms	(disk cache)
Steps.bf949cac48e7bb4d600c.png	200	fetch	serviceworker.js:24	0 B	7 ms	(disk cache)
faq3.93518a0b53a07dcaec14.png	200	fetch	serviceworker.js:24	0 B	7 ms	(disk cache)
collect?v=2&tid=G-SLDZGKJS3X&utm=45je5421v9205...	204	fetch	js?id=G-SLDZGKJS3X&utm=45je5421v9205...	0 B	80 ms	(ServiceWorker)
collect?v=2&tid=G-SLDZGKJS3X&utm=45je5421v92...	204	fetch	serviceworker.js:24	20 B	73 ms	

sync



Conclusion : Thus we learnt to implement service worker events like fetch, sync and push for PWA.