# EXPERIMENT 9

| **Name** | **Roll No.** |
|---|---|
| Anushka Shahane | **54** |
| Shravani Rasam | **45** |
| Pranav Titambe | **60** |

**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.');
        }
      }
    }
  });
```

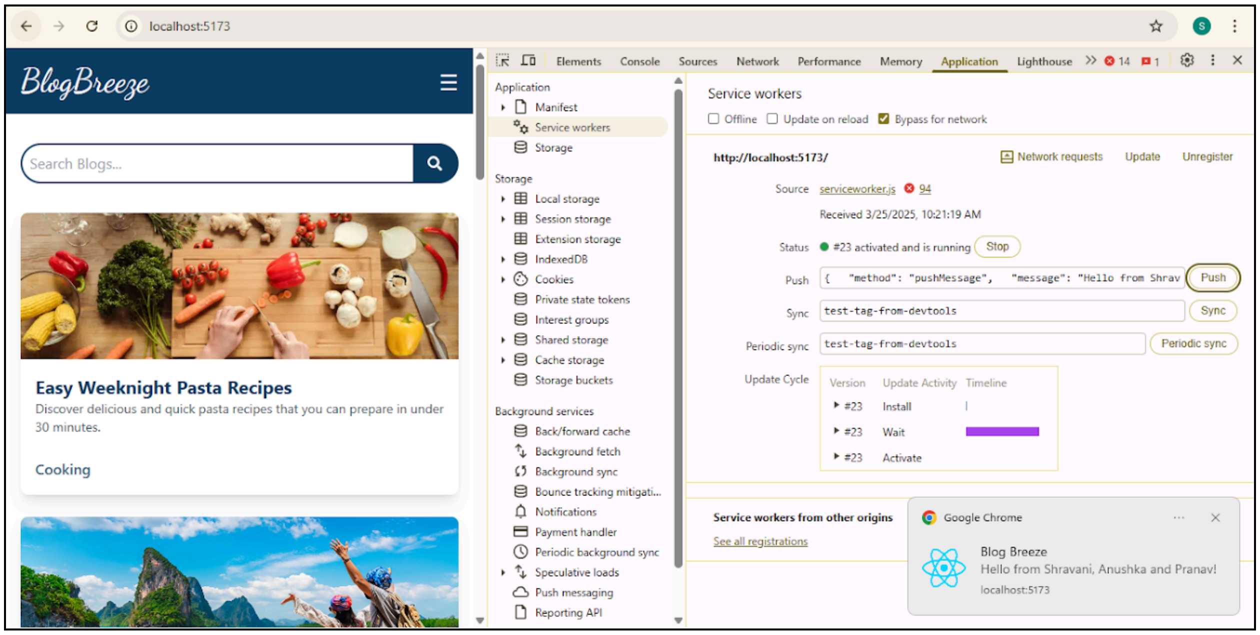
**SERVICE WORKER SCRIPT FOR SYNC AND FETCH :**

```
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 :



localhost:5173

BlogBreeze

Easy Weeknight Pasta Recipes

Discover delicious and quick pasta recipes that you can prepare in under 30 minutes.

Cooking

Service workers

Offline ☐ Update on reload ☐ Bypass for network ✓

http://localhost:5173/

Source serviceworker.js 94

Received 3/25/2025, 10:21:19 AM

Status #23 activated and is running

Push `{    "method": "pushMessage",    "message": "Hello from Shrav`

Sync `test-tag-from-devtools`

Periodic sync `test-tag-from-devtools`

Update Cycle

| Version | Update Activity | Timeline |
|---|---|---|
| #23 | Install | |
| #23 | Wait | |
| #23 | Activate | |

Service workers from other origins

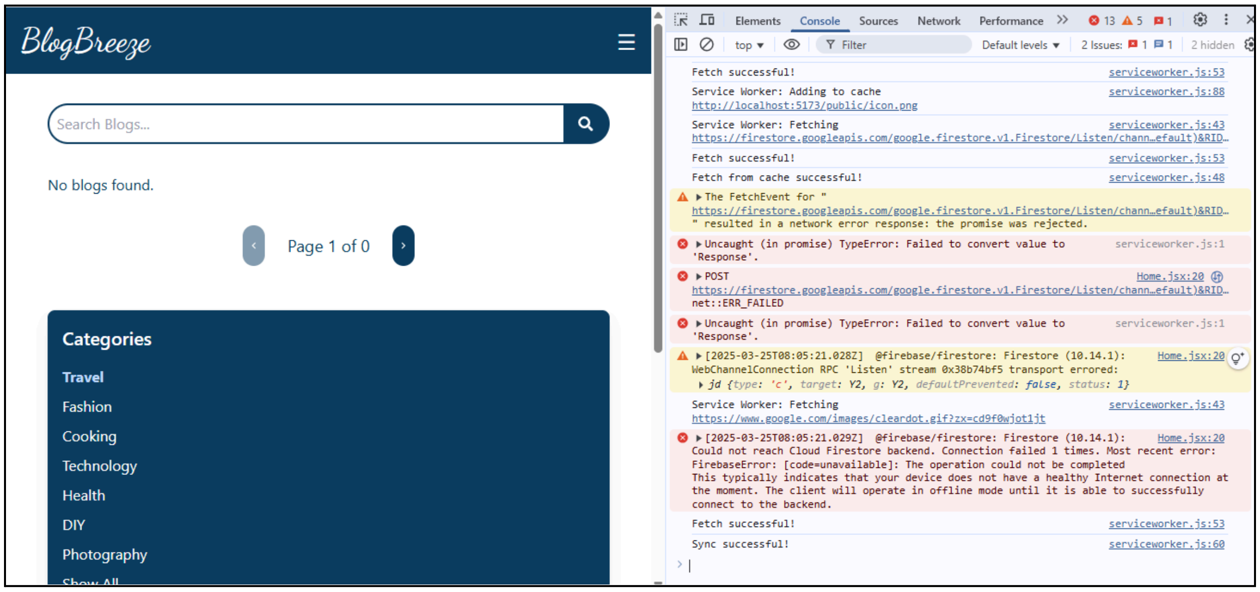
See all registrations

Google Chrome

Blog Breeze

Hello from Shravani, Anushka and Pranav!

localhost:5173



BlogBreeze

No blogs found.

Page 1 of 0

Categories

Travel

Fashion

Cooking

Technology

Health

DIY

Photography

```
Fetch successful!                                                    serviceworker.js:53
Service Worker: Adding to cache                                      serviceworker.js:88
http://localhost:5173/public/icon.png
Service Worker: Fetching                                             serviceworker.js:43
https://firestore.googleapis.com/google.firestore.v1.Firestore/Listen/chann…efault)&RID…
Fetch successful!                                                    serviceworker.js:53
Fetch from cache successful!                                         serviceworker.js:48
▶The FetchEvent for "
https://firestore.googleapis.com/google.firestore.v1.Firestore/Listen/chann…efault)&RID…
" resulted in a network error response: the promise was rejected.
▶Uncaught (in promise) TypeError: Failed to convert value to         serviceworker.js:1
'Response'.
▶POST                                                                Home.jsx:20
https://firestore.googleapis.com/google.firestore.v1.Firestore/Listen/chann…efault)&RID…
net::ERR_FAILED
▶Uncaught (in promise) TypeError: Failed to convert value to         serviceworker.js:1
'Response'.
▶[2025-03-25T08:05:21.028Z]  @firebase/firestore: Firestore (10.14.1):   Home.jsx:20
WebChannelConnection RPC 'Listen' stream 0x38b74bf5 transport errored:
 ▶jd {type: 'c', target: Y2, g: Y2, defaultPrevented: false, status: 1}
Service Worker: Fetching                                             serviceworker.js:43
https://www.google.com/images/cleardot.gif?zx=cd9f0wjot1jt
▶[2025-03-25T08:05:21.029Z]  @firebase/firestore: Firestore (10.14.1):   Home.jsx:20
Could not reach Cloud Firestore backend. Connection failed 1 times. Most recent error:
FirebaseError: [code=unavailable]: The operation could not be completed
This typically indicates that your device does not have a healthy Internet connection at
the moment. The client will operate in offline mode until it is able to successfully
connect to the backend.
Fetch successful!                                                    serviceworker.js:53
Sync successful!                                                     serviceworker.js:60
```

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