

Experiment 9

Aim- To implement Service worker events like fetch, sync and push for Nwes-App PWA.

Theory:

Service Worker:-

A Service Worker is a script that runs in the background of a web application, separate from the main browser thread. It enables powerful features like offline support, background sync, and push notifications by intercepting network requests and caching resources. Service workers are essential for building Progressive Web Apps (PWAs), allowing them to function reliably even with limited or no internet connectivity. They follow a lifecycle of installation, activation, and fetching, and must be served over HTTPS for security reasons. Overall, service workers enhance performance, reliability, and user experience in modern web applications.

Fetch Event:-

The fetch event in a service worker is triggered whenever the web app makes a network request, such as fetching a file, API data, or image. It allows the service worker to intercept these requests and respond with cached content or fetch fresh data from the network. By handling fetch events, developers can implement offline support and faster load times by serving resources from the cache. The `self.addEventListener('fetch', ...)` method is used to listen for and manage these events. This event plays a crucial role in Progressive Web Apps by improving performance and reliability. It also allows for custom request handling, such as fallback pages or dynamic caching.

Sync Event:-

The sync event in a service worker is triggered when the browser regains internet connectivity after being offline. It allows the service worker to retry failed actions, such as sending data to a server or syncing content. This is especially useful for background tasks like posting messages, updating content, or syncing user activity that couldn't complete earlier. The sync event is handled using

`self.addEventListener('sync', ...)`, and it typically works in combination with the Background Sync API. By using sync events, developers can ensure a smoother and more reliable user experience, even with intermittent connectivity. It helps maintain data consistency and enhances the overall functionality of Progressive Web Apps.

Push Event:-

The push event in a service worker is triggered when a push message is received from a server, even if the web app is not currently open. It enables web apps to receive and display notifications in real-time, improving user engagement. The service worker listens for this event using `self.addEventListener('push', ...)` and can show notifications using the `showNotification()` method. Push events are commonly used for updates, alerts, messages, and reminders. They rely on the Push API and require user permission to receive notifications. This feature allows Progressive Web Apps to deliver timely information and stay connected with users even when the app is not actively running.

Code:

Fetch Event:-

// Fetch event

```
self.addEventListener('fetch', event => {
  if (event.request.method !== 'GET') return;

  event.respondWith(
    caches.match(event.request).then(cached => {
      if (cached) {
        console.log('[Service Worker] Serving from cache:', event.request.url);
        return cached;
      }

      return fetch(event.request)
        .then(response => {
          return caches.open(CACHE_NAME).then(cache => {
            cache.put(event.request, response.clone());
            console.log('[Service Worker] Fetched and cached:', event.request.url);
```

```

        return response;
    });
})
.catch(() => {
    console.log('[Service Worker] Fetch failed, serving offline page.');
```

return caches.match('/offline.html');

```

    });
})
);
});
```


Sync Event:-

```

// Sync event
self.addEventListener('sync', event => {
    console.log('[Service Worker] Sync event fired with tag:', event.tag);
    if (event.tag === 'sync-news') {
        event.waitUntil(sendQueuedArticles());
    }
});
```

Push Event:-

```

//  Push event
self.addEventListener('push', event => {
    if (event.data) {
        const data = event.data.json();
        if (data.method === 'pushMessage') {
            const options = {
                body: data.message,
                icon: '/icons/icon-192x192.png',
                badge: '/icons/icon-192x192.png'
            };
            event.waitUntil(
                self.registration.showNotification("News App", options)
            );
        }
    }
});
```

```
}  
});
```

Github Link:-

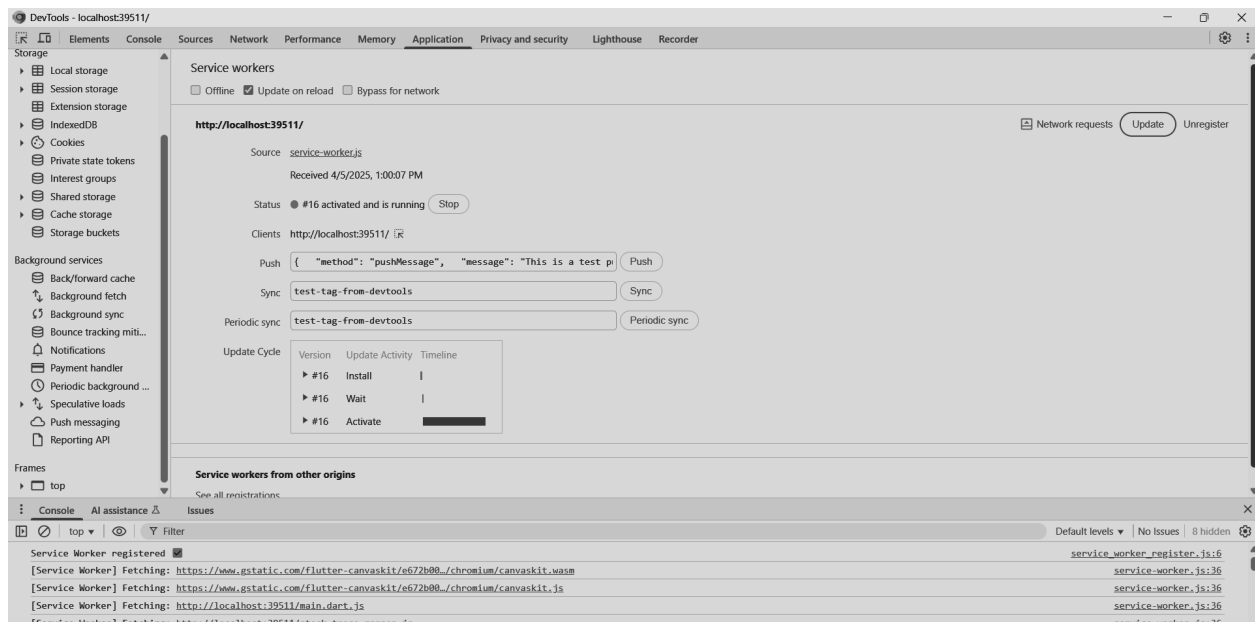
https://github.com/Ishan2611/News_Edge/commit/370e22a81895ea509fbc23f5b7b466d3d4728d52

Output:

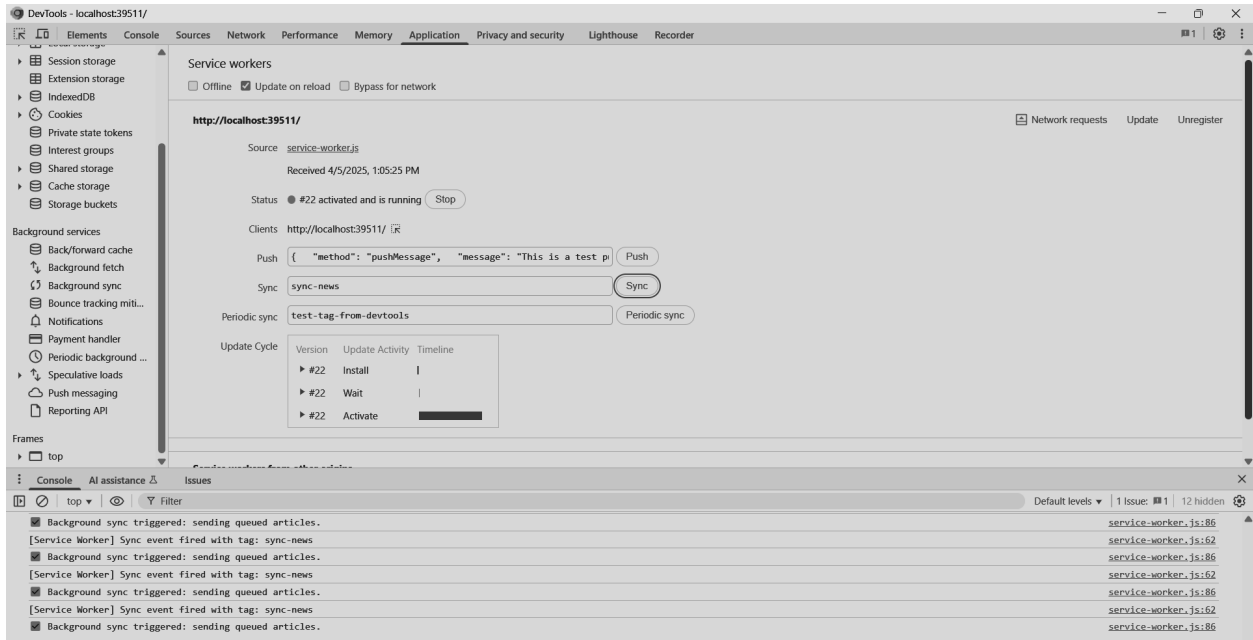
1-Fetch Event

2-Sync Event

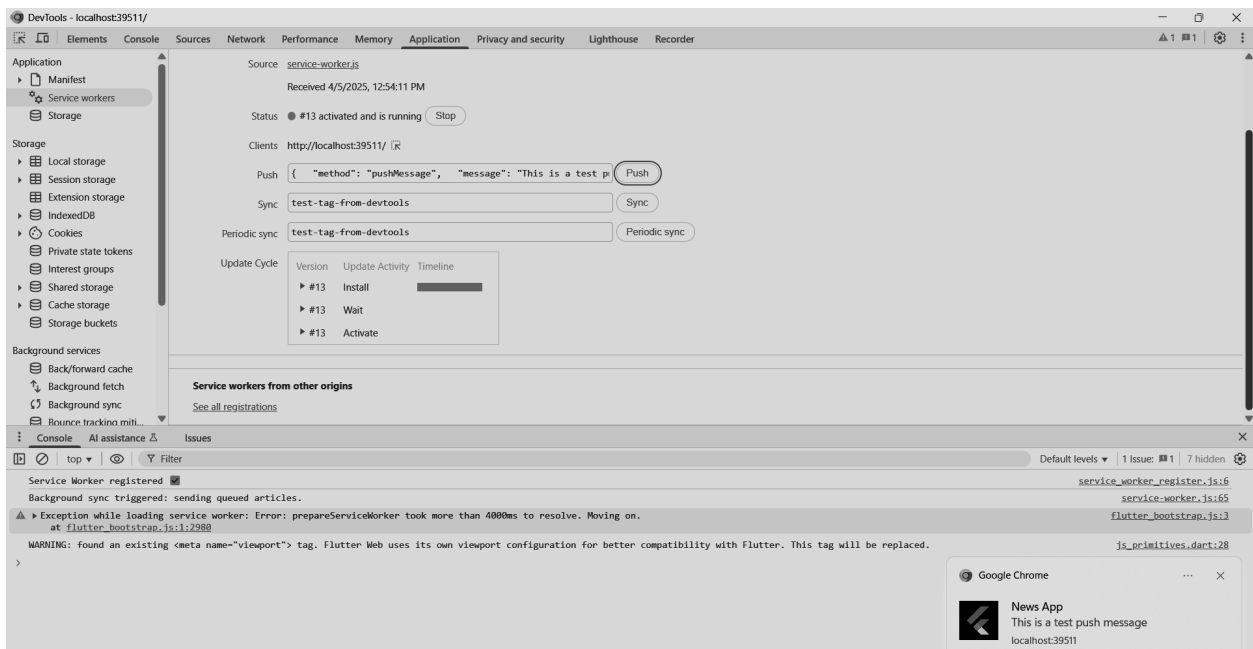
3-Push Event



The fetch event in a service worker is triggered whenever the application makes a network request, such as loading data or assets. It allows the service worker to intercept these requests and respond with cached content or fetch it from the network. This feature is essential for building Progressive Web Apps (PWAs) that work offline or load faster. In the image, the console shows fetch events being logged, indicating the service worker is actively handling requests. Overall, the fetch event improves performance and reliability by managing how resources are loaded.



This displays the sync event feature in the Service Workers panel of Chrome DevTools. The sync event is used in service workers to handle background synchronization when the device regains internet connectivity. In the image, a sync tag named "sync-news" is triggered, and the console logs show that the service worker is sending queued articles. This demonstrates how sync events can ensure data is sent or updated even after being offline. Overall, the sync event improves reliability by allowing tasks like sending form data or updates to complete once the network is available again.



This shows a push event being triggered in the Service Workers panel of Chrome DevTools. A test push message with the content "This is a test push message" is sent, and a notification appears on the screen from the News App. Push events allow service workers to receive and display messages even when the web app is not active. This feature is useful for sending real-time updates or alerts to users.

Conclusion:

In this experiment, we successfully implemented key service worker events—fetch, sync, and push—to enhance the functionality of the News-App PWA. The fetch event enabled offline support by serving cached content, improving load times and reliability. The sync event allowed background data synchronization when the connection was restored, ensuring seamless user experience even with intermittent connectivity. The push event enabled real-time notifications, keeping users informed and engaged even when the app was not active. Together, these events demonstrated the power of service workers in creating responsive, reliable, and user-friendly Progressive Web Apps.