

EXPERIMENT NO. 9

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.

```

self.addEventListener("fetch", function (event) {
  const req = event.request;
  const url = new URL(req.url);

  if (url.origin === location.origin) {
    event.respondWith(cacheFirst(req));
  }
  else {
    event.respondWith(networkFirst(req));
  }
});

async function cacheFirst(req) {
  return await caches.match(req) || fetch(req);
}

async function networkFirst(req) {
  const cache = await caches.open("pwa-dynamic");
  try {
    const res = await fetch(req);
    cache.put(req, res.clone());
    return res;
  } catch (error) {
    const cachedResponse = await cache.match(req);
    return cachedResponse || await caches.match("./noconnection.json");
  }
}

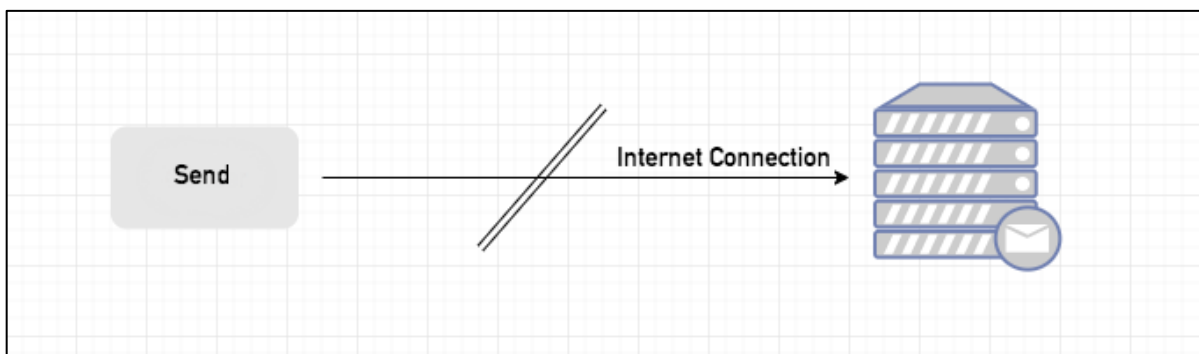
```

Sync Event

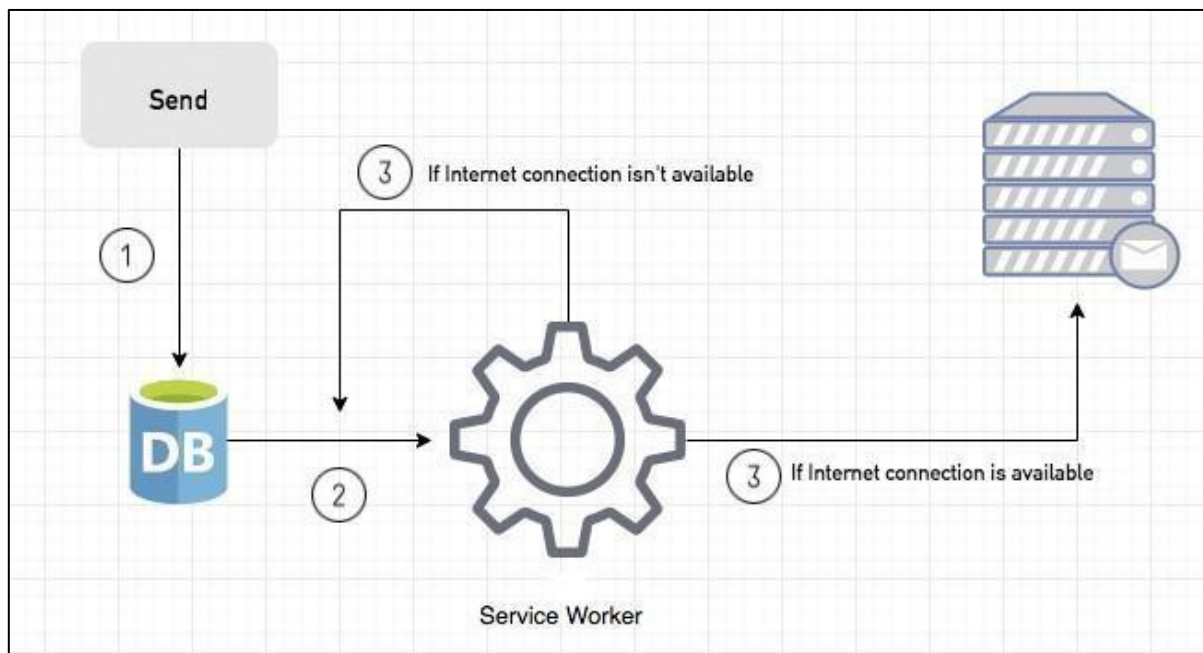
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are 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.



1. When we click the “send” button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **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

```
document.querySelector("button").addEventListener("click", async () => {
  var swRegistration = await navigator.serviceWorker.register("sw.js");
  swRegistration.sync.register("helloSync").then(function () {
    console.log("helloSync success [main.js]");
  });
});
```

Event Listener for sw.js

```
self.addEventListener('sync', event => {
  if (event.tag == 'helloSync') {
    console.log("helloSync [sw.js]");
  }
});
```

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.

```
self.addEventListener('push', event => {  
  if (event && event.data) {  
    var data = event.data.json();  
    if (data.method === "pushMessage") {  
      event.waitUntil(self.registration.showNotification("Test App", {  
        body: data.message  
      }));  
    }  
  }  
});
```

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

Code:**1. Push Notification code:**

```
/ Push Notification event - College Announcements
self.addEventListener("push", async (event) => {
  if (!self.registration || Notification.permission !== "granted") {
    console.warn("Push notification permission not granted.");
    return;
  }

  const options = {
    body: "New event update at VESIT! Click to know more.",
    icon: "/logo192.png",
    badge: "/logo192.png",
    actions: [
      { action: "view", title: "View Event" },
      { action: "close", title: "Dismiss" }
    ]
  };

  event.waitUntil(
    self.registration.showNotification("College Announcement!", options)
      .catch(error => console.error("Error showing notification:", error))
  );
});

// Handle Notification Click - Redirect to Event Page
self.addEventListener("notificationclick", (event) => {
  event.notification.close();

  if (event.action === "view") {
    event.waitUntil(
      clients.matchAll({ type: "window", includeUncontrolled: true }).then((clientList) => {
        if (clientList.length > 0) {
          return clientList[0].focus();
        }
        return clients.openWindow("/events").catch((error) => {
          console.error("Failed to open events page:", error);
        });
      })
    );
  }
});
```

2. Fetch and Sync Code:

```
// Fetch event - Serve files from cache & dynamically cache new requests
self.addEventListener("fetch", (event) => {
  if (event.request.method !== "GET") return; // Handle only GET requests

  event.respondWith(
    caches.match(event.request).then((cachedResponse) => {
      if (cachedResponse) {
        return cachedResponse; // Return cached response if available
      }

      return fetch(event.request)
        .then((fetchResponse) => {
          return caches.open(CACHE_NAME).then((cache) => {
            cache.put(event.request, fetchResponse.clone());
            return fetchResponse;
          });
        })
        .catch(() => {
          return new Response("You are offline. Please check your connection.", {
            status: 503,
            headers: { "Content-Type": "text/plain" }
          });
        });
    })
  );
});

// Background Sync - Preload Important Pages
self.addEventListener("sync", (event) => {
  if (event.tag === "preload-pages") {
    event.waitUntil(preloadImportantPages());
  }
});

async function preloadImportantPages() {
  console.log("Preloading important pages...");
  const pages = ["/courses", "/teachers", "/about"];
  try {
    const cache = await caches.open(CACHE_NAME);
    await Promise.all(pages.map(async (page) => {
      const response = await fetch(page);
      if (response.ok) {
        cache.put(page, response.clone());
      }
    }));
    console.log("Pages preloaded successfully.");
  } catch (error) {
    console.error("Preloading failed:", error);
  }
}
```

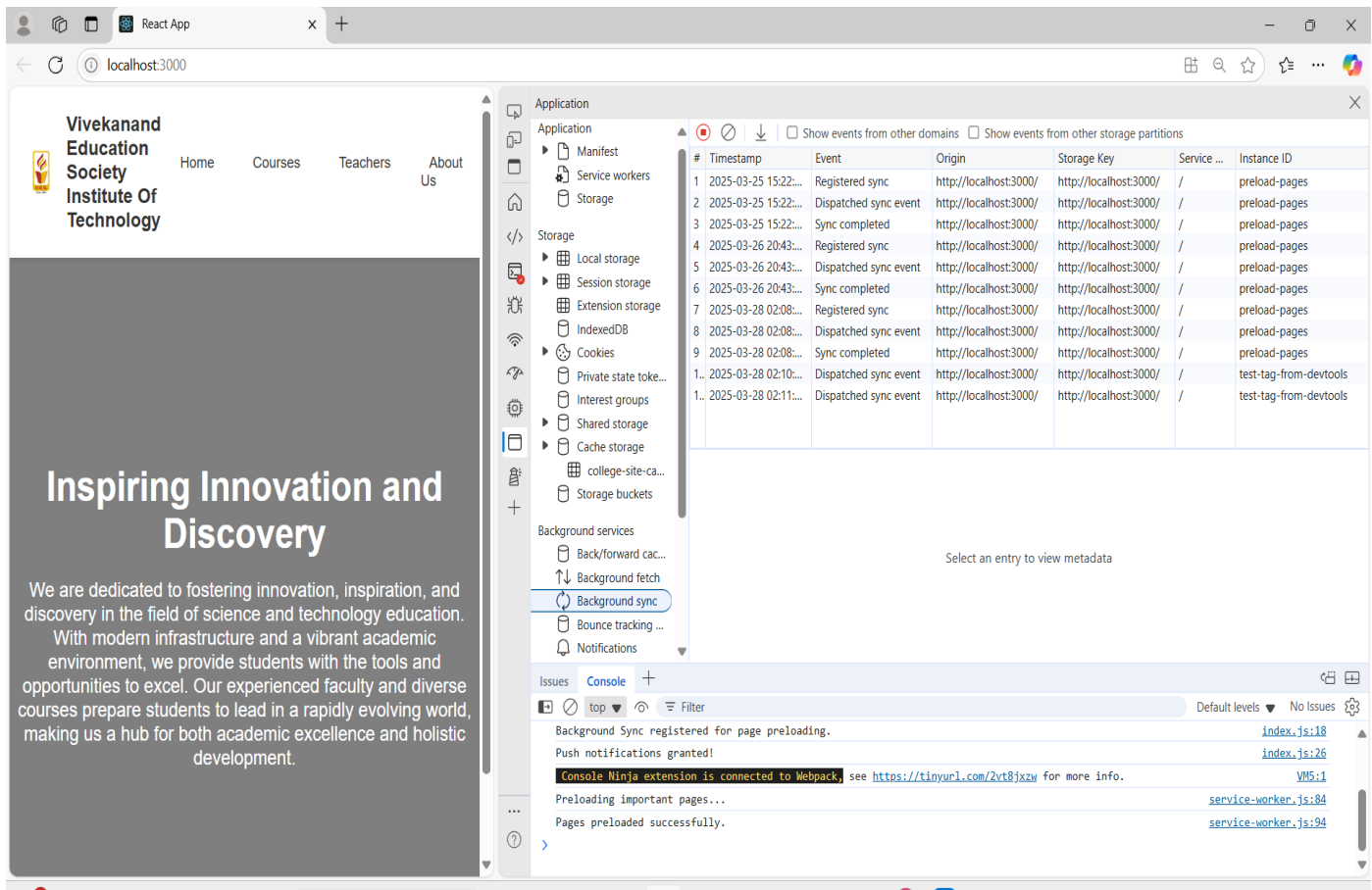
OUTPUT:-

The screenshot shows a web browser at localhost:3000 displaying the Vivekanand Education Society Institute of Technology website. The website has a navigation bar with links to Home, Courses, Teachers, and About Us. The main content area features the text "Inspiring Innovation and Discovery" and a paragraph about the institute's dedication to fostering innovation and discovery in science and technology education.

Overlaid on the browser is the Chrome DevTools Service Workers panel. The panel shows the service worker for http://localhost:3000/ with the source service-worker.js. It indicates that the service worker is activated and running. The panel also shows the update cycle and a list of clients. A notification bubble is visible in the bottom right corner of the browser window, stating "College Announcement! New event update at VESIT! Click to know more. via Microsoft Edge" with buttons for "View Event" and "Dismiss".

The screenshot shows the Chrome DevTools Storage panel, specifically the "Push messaging" section. The panel displays a list of push messaging events with columns for #, Timestamp, Event, Origin, Storage Key, Service, and Instance ID. The events are listed in chronological order, showing the sequence of push messages received by the service worker.

#	Timestamp	Event	Origin	Storage Key	Service	Instance ID
1.	2025-03-28 02:0...	Push event dispatc...	http://localhost:30...	http://localhost:30...	/	
2.	2025-03-28 02:0...	Push event comple...	http://localhost:30...	http://localhost:30...	/	
3.	2025-03-28 02:0...	Push event dispatc...	http://localhost:30...	http://localhost:30...	/	
4.	2025-03-28 02:0...	Push event comple...	http://localhost:30...	http://localhost:30...	/	
5.	2025-03-28 02:0...	Push event dispatc...	http://localhost:30...	http://localhost:30...	/	
6.	2025-03-28 02:0...	Push event comple...	http://localhost:30...	http://localhost:30...	/	
7.	2025-03-28 02:1...	Push event dispatc...	http://localhost:30...	http://localhost:30...	/	
8.	2025-03-28 02:1...	Push event comple...	http://localhost:30...	http://localhost:30...	/	
9.	2025-03-28 02:1...	Push event dispatc...	http://localhost:30...	http://localhost:30...	/	
1.	2025-03-28 02:1...	Push event comple...	http://localhost:30...	http://localhost:30...	/	



Vivekanand Education Society Institute Of Technology

Home Courses Teachers About Us

Inspiring Innovation and Discovery

We are dedicated to fostering innovation, inspiration, and discovery in the field of science and technology education. With modern infrastructure and a vibrant academic environment, we provide students with the tools and opportunities to excel. Our experienced faculty and diverse courses prepare students to lead in a rapidly evolving world, making us a hub for both academic excellence and holistic development.

Application

Application

- Manifest
- Service workers
- Storage

Storage

- Local storage
- Session storage
- Extension storage
- IndexedDB
- Cookies
- Private state token
- Interest groups
- Shared storage
- Cache storage
- college-site-cache
- Storage buckets

Background services

- Back/forward cache
- Background fetch
- Background sync**
- Bounce tracking ...
- Notifications

Events

#	Timestamp	Event	Origin	Storage Key	Service ...	Instance ID
1	2025-03-25 15:22:...	Registered sync	http://localhost:3000/	http://localhost:3000/	/	preload-pages
2	2025-03-25 15:22:...	Dispatched sync event	http://localhost:3000/	http://localhost:3000/	/	preload-pages
3	2025-03-25 15:22:...	Sync completed	http://localhost:3000/	http://localhost:3000/	/	preload-pages
4	2025-03-26 20:43:...	Registered sync	http://localhost:3000/	http://localhost:3000/	/	preload-pages
5	2025-03-26 20:43:...	Dispatched sync event	http://localhost:3000/	http://localhost:3000/	/	preload-pages
6	2025-03-26 20:43:...	Sync completed	http://localhost:3000/	http://localhost:3000/	/	preload-pages
7	2025-03-28 02:08:...	Registered sync	http://localhost:3000/	http://localhost:3000/	/	preload-pages
8	2025-03-28 02:08:...	Dispatched sync event	http://localhost:3000/	http://localhost:3000/	/	preload-pages
9	2025-03-28 02:08:...	Sync completed	http://localhost:3000/	http://localhost:3000/	/	preload-pages
1..	2025-03-28 02:10:...	Dispatched sync event	http://localhost:3000/	http://localhost:3000/	/	test-tag-from-devtools
1..	2025-03-28 02:11:...	Dispatched sync event	http://localhost:3000/	http://localhost:3000/	/	test-tag-from-devtools

Select an entry to view metadata

Issues **Console**

Background Sync registered for page preloading. [index.js:18](#)

Push notifications granted! [index.js:26](#)

Console Ninja extension is connected to Webpack, see <https://tinyurl.com/2vt8jxzw> for more info. [VM5:1](#)

Preloading important pages... [service-worker.js:84](#)

Pages preloaded successfully. [service-worker.js:94](#)