

# 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.	14
Name	Koml Milind Deolekar
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 Website 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");
  }
}

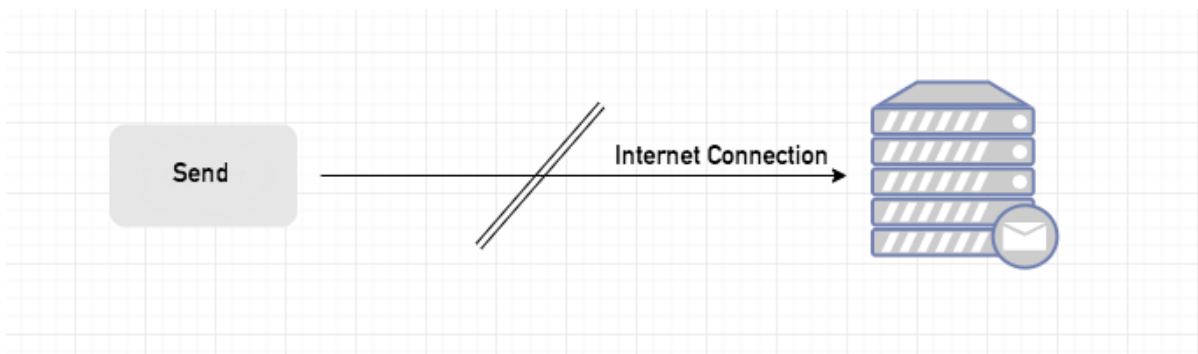
```

### Sync Event

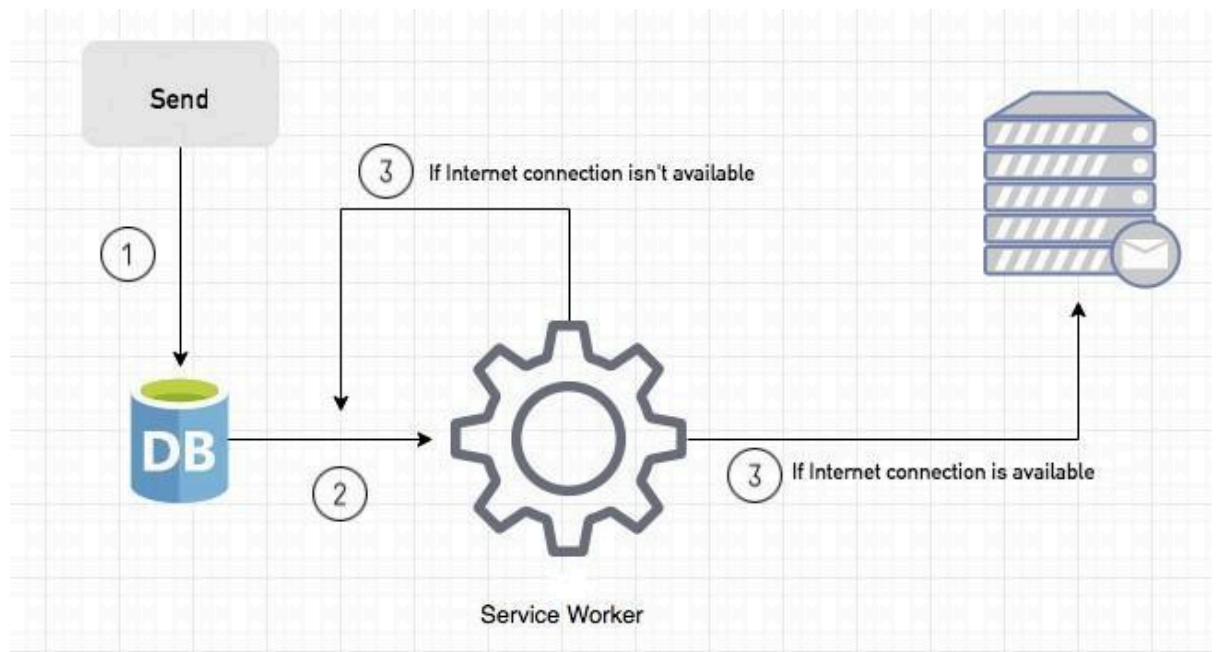
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

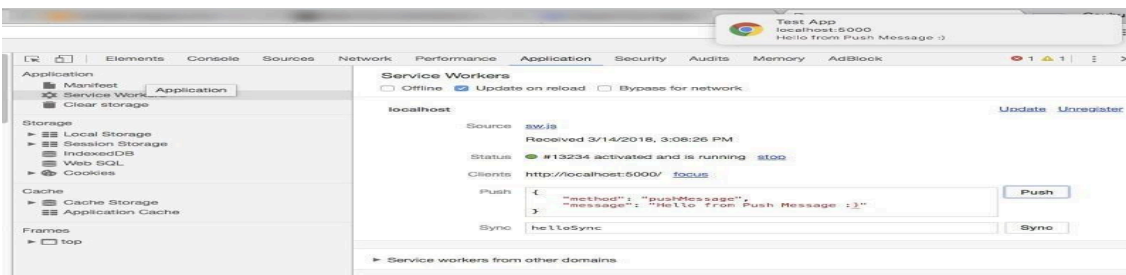
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

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

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

value is “pushMessage”, we open the information notification with the “message” property.

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



## Code :

### serviceworker.js

```
const cacheName = 'vesit-pwa-v1';
const assetsToCache = [
  '/',
  '/manifest.json',
  '/vite.svg',
  '/vesitlogo.jpg',
  '/screenshot1.png',
];

// ✅ Install
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(cacheName).then(cache => cache.addAll(assetsToCache))
  );
  self.skipWaiting();
});

// ✅ Activate
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(
        keys.filter(name => name !== cacheName).map(name => caches.delete(name))
      );
    })
  );
  self.clients.claim();
});


// ✅ Fetch (Offline support + Notification on failure)
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request).catch(() => {
        showNotification('Network Error', 'Failed to fetch: ' + event.request.url);
        return new Response('Offline', {
          status: 408,
          statusText: 'Offline'
        });
      });
    })
  );
});

// ✅ Sync (send data + show notification on success/failure)
self.addEventListener('sync', event => {
  if (event.tag === 'sync-feedback') {
    event.waitUntil(
      sendFeedbackData()
    );
  }
});

async function sendFeedbackData() {
  try {
    const offlineData = await getOfflineFeedback();
    await fetch('/api/submit-feedback', {
```


```
    method: 'POST',
    body: JSON.stringify(offlineData),
    headers: {
      'Content-Type': 'application/json'
    }
  });
  showNotification('Feedback Synced', 'Your feedback was sent successfully!');
} catch (err) {
  showNotification('Sync Failed', 'Could not sync feedback.');
  console.error('Sync failed:', err);
}
}
```

```
// Example mock data function
function getOfflineFeedback() {
  return Promise.resolve({
    name: 'Komal Deolekar',
    message: 'Great college app!',
    timestamp: new Date().toISOString()
  });
}
```

```
//  Push (show incoming push notification)
```

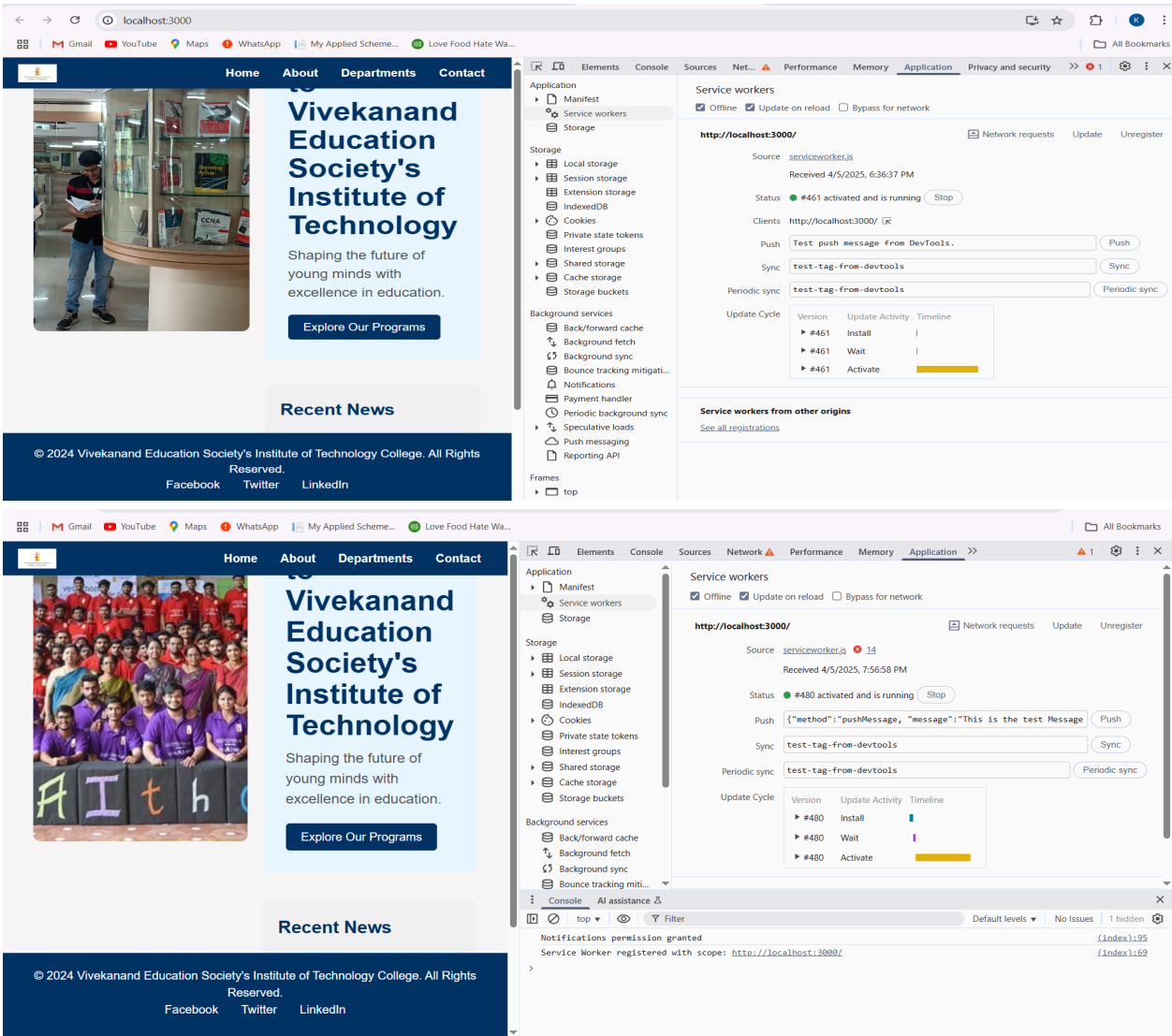
```
self.addEventListener('push', event => {
  const data = event.data?.json() || {
    title: 'VESIT Update',
    body: 'Admissions open for 2025!',
    icon: '/vite.svg'
  };

  event.waitUntil(
    self.registration.showNotification(data.title, {
      body: data.body,
      icon: data.icon
    })
  );
});
```

```
//  Helper: Show notification (reusable across all)
```

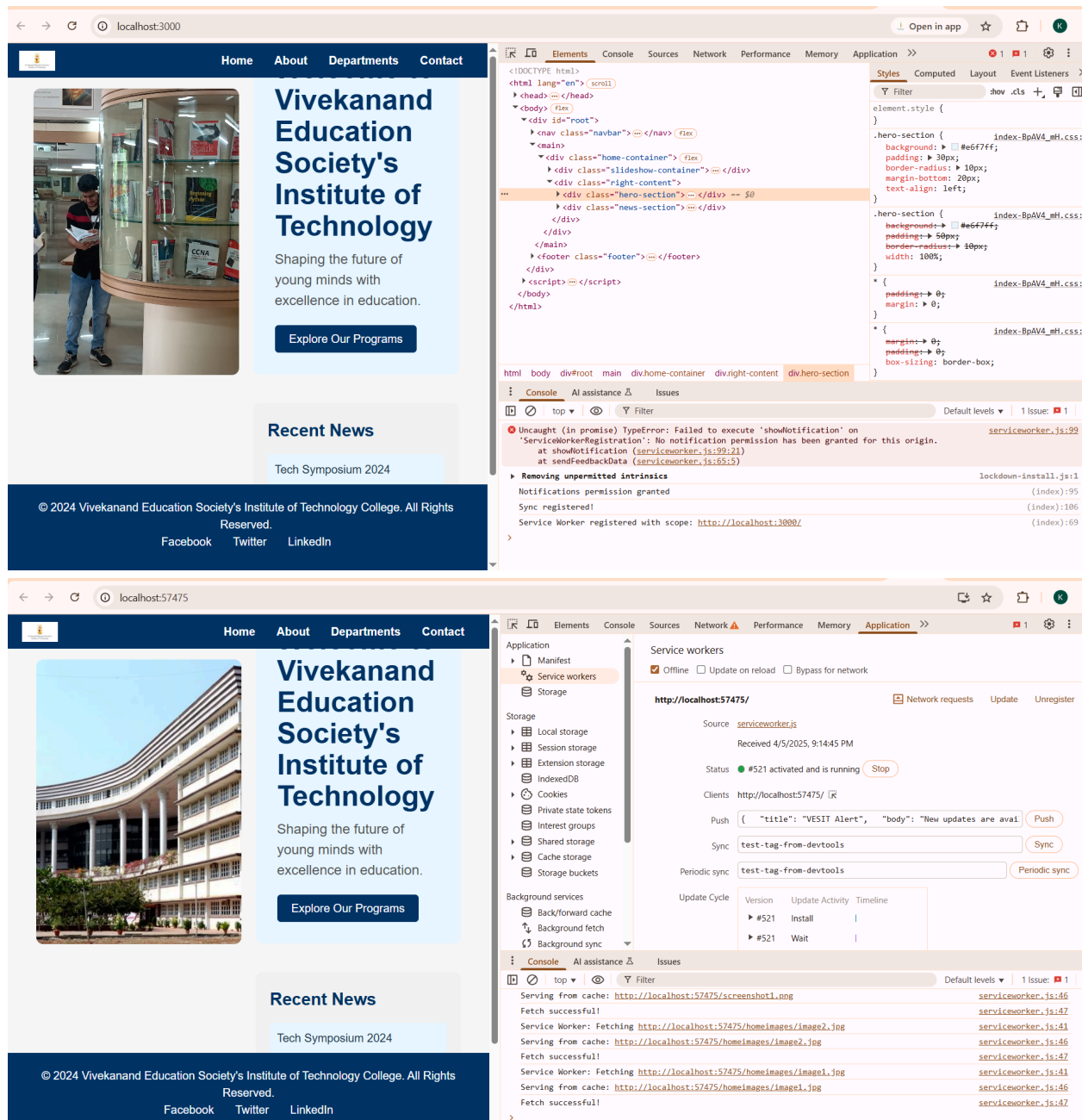
```
function showNotification(title, body) {
  self.registration.showNotification(title, {
    body,
    icon: '/vite.svg'
  });
}
```

Output :






## Fetch Event



Sync Event




Home

About

Departments

Contact



# Vivekanand Education Society's Institute of Technology

Shaping the future of young minds with excellence in education.

Explore Our Programs

Recent News

Tech Symposium 2024

© 2024 Vivekanand Education Society's Institute of Technology College. All Rights Reserved.

Facebook

Twitter

LinkedIn

Service workers

Offline

Update on reload

Bypass for network

http://localhost:3000/

Network requests

Update

Unregister

Source

serviceworker.js

Received 4/5/2025, 7:55:58 PM

Status

#479 activated and is running

Stop

Push

{ "method": "pushMessage", "message": "This is the test Message" }

Push

Sync

test-tag-from-devtools

Sync

Periodic sync

test-tag-from-devtools

Periodic sync

Update Cycle

Version	Update Activity	Timeline
#479	Install	
#479	Wait	
#479	Activate	

Notifications permission granted

Feedback synced successfully!

Service Worker registered with scope: http://localhost:3000/

localhost:3000

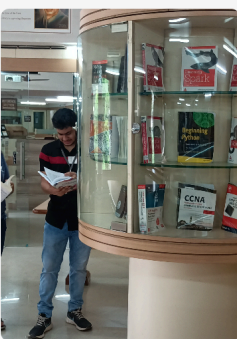
Open in app

Home

About

Departments

Contact



# Vivekanand Education Society's Institute of Technology

Shaping the future of young minds with excellence in education.

Explore Our Programs

Recent News

Tech Symposium 2024

© 2024 Vivekanand Education Society's Institute of Technology College. All Rights Reserved.

Facebook

Twitter

LinkedIn

Elements

Console

Sources

Network

Performance

Memory

Application

<!DOCTYPE html>

<html lang="en">

<head>

</head>

<body>

<div id="root">

<div class="navban">

<div class="home-container">

<div class="slideshow-container">

<div class="right-content">

<div class="hero-section">

<div class="news-section">

</div>

</main>

</div>

<div class="footer">

</div>

<script>

</script>

</body>

</html>

Styles

Computed

Layout

Event Listeners

element.style

.hero-section

.hero-section

\* {

padding:

margin:

\* {

margin:

padding:

box-sizing:

Uncaught (in promise) TypeError: failed to execute 'showNotification' on 'ServiceWorkerRegistration': No notification permission has been granted for this origin.

at showNotification (serviceworker.js:99:21)

at sendFeedbackData (serviceworker.js:65:5)

Removing unpermitted intrinsics

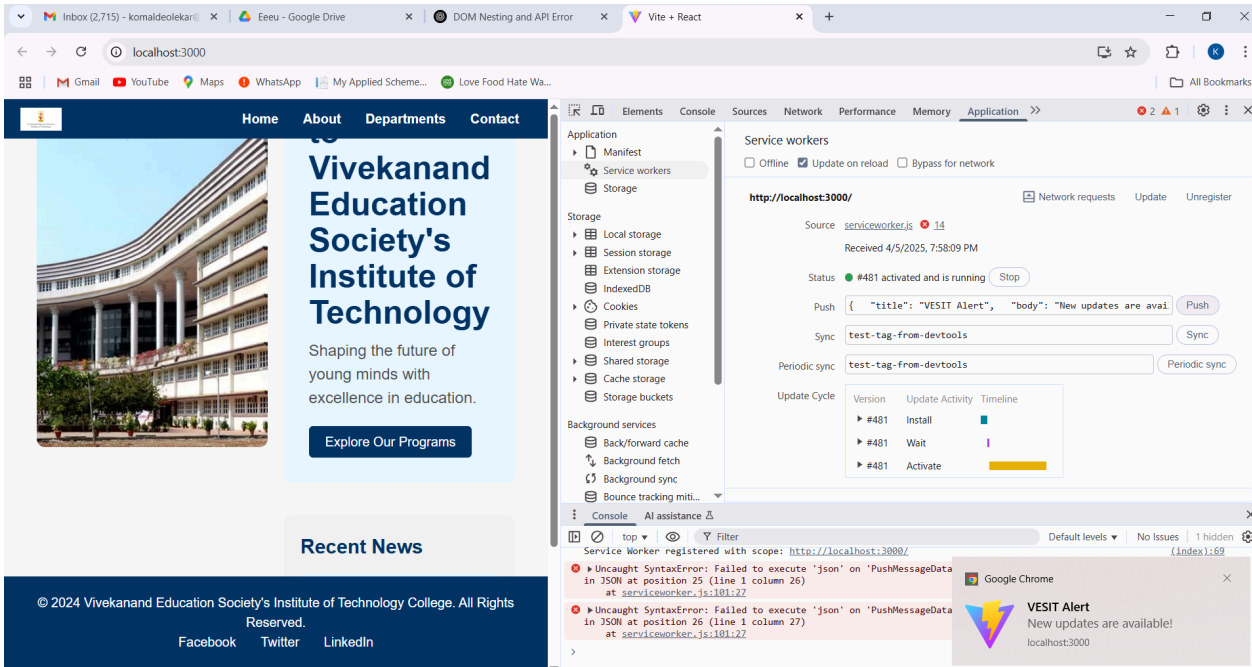
lockdown-install.js:1

Notifications permission granted

Sync registered!

Service Worker registered with scope: http://localhost:3000/

Push event



Conclusion :

By implementing service worker events like **fetch**, **sync**, and **push**, we enhance the Website PWA with offline access, background data synchronization, and real-time notifications. This ensures improved performance, reliability, and user engagement even in low or no network conditions.



