Experiment No.9
MAD & PWA LAB

- **Aim:** To implement Service worker events like fetch, sync and push for E-commerce PWA.

- **Theory:**

→ Service-Worker –

A service worker is a script that runs independently in the browser background. On the user side, it can intercept its network requests and decide what to load (fetch).
Service workers mainly serve features like background sync, push notifications and they are commonly used for'offline first' applications, giving the developers the opportunity to take complete control over the user experience.

Before it's time there has been API called AppCache, which has been trying to serve the offline experience feature. However, there have been numerous problems in the interface of the AppCache API and Service Workers are here, going over them.

→ The service worker life cycle –

The service worker lifecycle is completely separate from the web page. It's a programmable network proxy, which is terminated when it's not used and restarted when it's next needed. Service Workers heavily rely on the use of Javascript Promises , so it's good to go over them if they are new to you.

During installation, the service worker can cache some static assets like web pages. If the browser cache the files successfully, the service worker gets installed.

Afterward, the worker needs to be activated. During activation the service worker can manage and decide what to happen to the old caches, typically they are being deleted and replaced with the new versions.

Lastly, after activation, the service worker takes control over all pages in his scope, without the page which initially registered the service worker, which needs to be refreshed. The service worker is smart in terms of memory usage and will be terminated if there is nothing to fetch and there are no message events occurring.

→ Events :

[1] Fetch Event:

- The fetch event triggers whenever the service worker intercepts a network request initiated by the web page. It grants the service worker the ability to intercept and manipulate network requests, enabling the implementation of various caching strategies and offline functionality.
- This event is commonly employed to execute caching strategies such as Cache First or Network First. These strategies empower the service worker to respond with cached assets when the user is offline or when network connectivity is poor.
- For instance, in the provided service worker code, the fetch event is utilized to cache fetched files within the 'v1' cache and to serve them from the cache when available.

[2] Sync Event:

- The sync event occurs when the browser initiates a background synchronization with the service worker. This feature enables the service worker to execute tasks, like sending data to a server, even when the application is not actively engaged or when the device lacks network connectivity.
- The sync event proves beneficial for tasks necessitating periodic updates or data synchronization, such as synchronizing offline data with a server once network connectivity is restored.

[3] Push Event:

- The push event is triggered when the service worker receives a push notification from a push service. It enables the service worker to manage the push notification, which may involve displaying a notification to the user or executing background tasks.
- Typically, the push event is employed to inform users about specific events or updates through notifications, such as new messages or app updates.
- Within the given service worker code, the push event is integrated to handle incoming push notifications. It verifies whether the push message method is "pushMessage" and subsequently presents a notification containing the message content to the user.

- **Implementation:**

→ **fetch function[code] –**

```
self.addEventListener('fetch', event => {
  console.log('Service Worker: Fetching');
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        if (response) {
          console.log('Service Worker: Found in Cache');
          return response;
        }
        return fetch(event.request)
          .then(response => {
            if (!response || response.status !== 200 ||
                response.type !== 'basic') {
              return response;
            }
            const responseToCache = response.clone();
            caches.open(CACHE_NAME)
              .then(cache => {
                cache.put(event.request, responseToCache);
              });
            return response;
          });
      })
  );
});
```

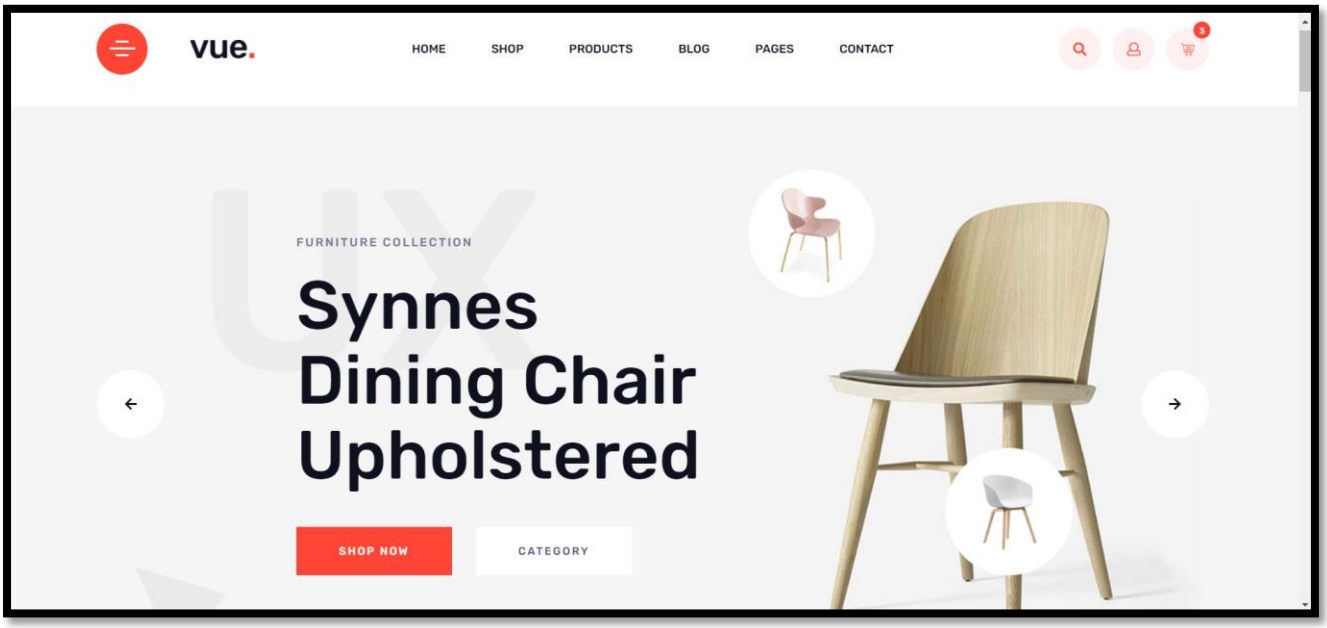
→ **sync function [code] –**

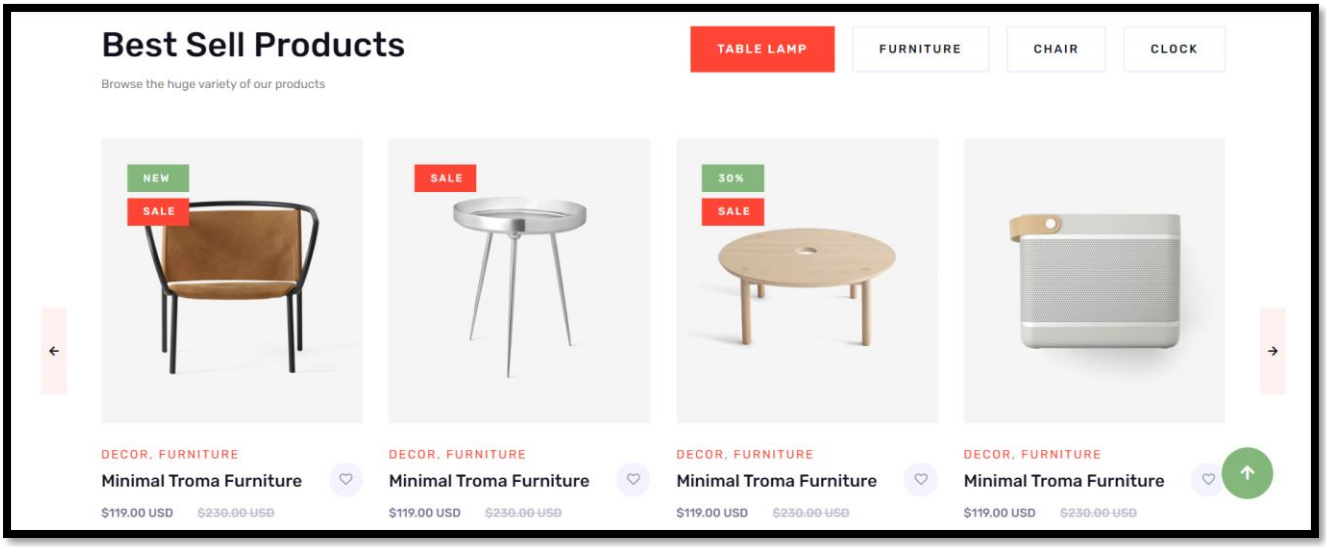
```
self.addEventListener('sync', event => {
  if (event.tag === 'sync-data') {
    console.log('Service Worker: Sync event triggered');
  }});
```

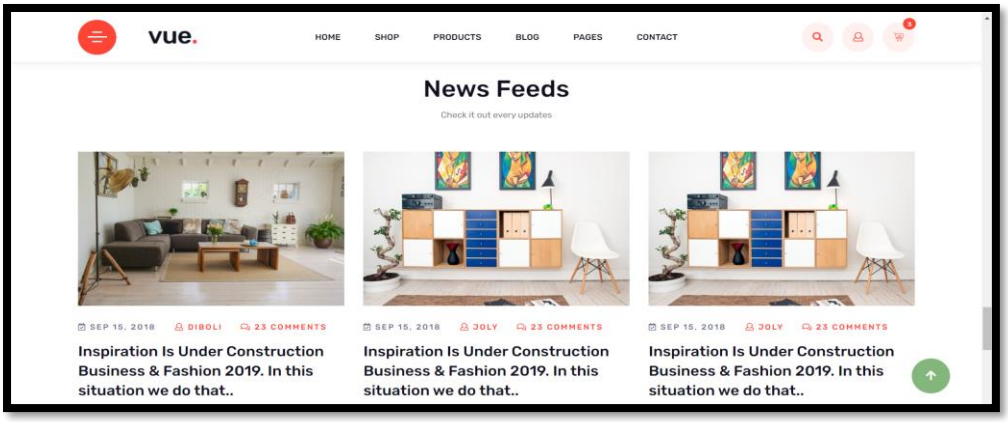
→ **push function [code] –**

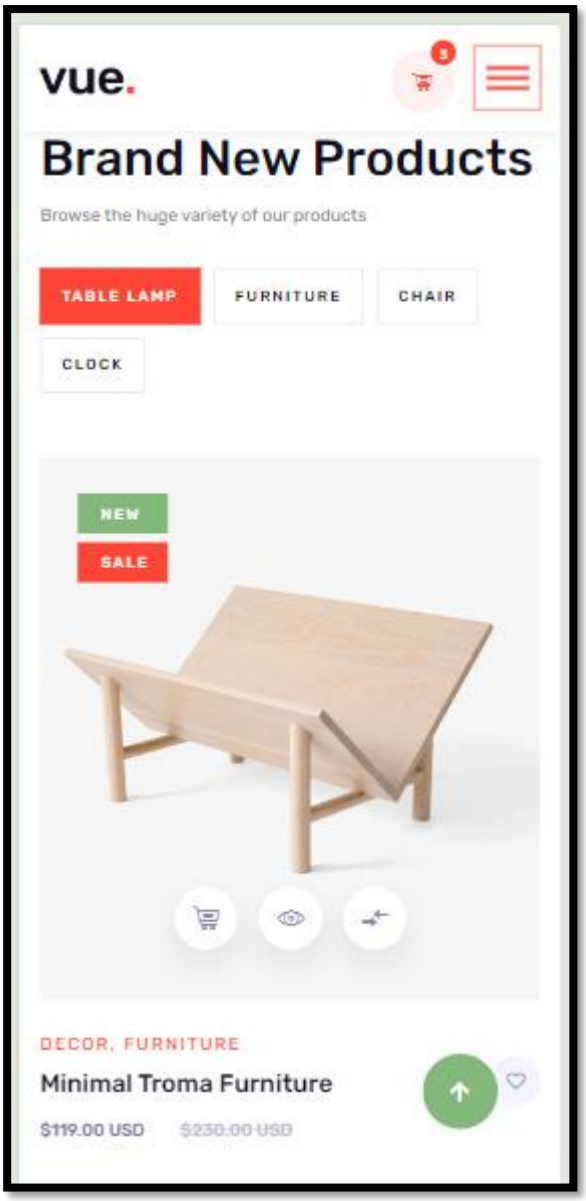
```
self.addEventListener('push', event => {
    console.log('Service Worker: Push event received', event);
  const title = 'E-commerce PWA';
  const options = {
    body: 'New content is available!',
    icon: 'icon.png',
    badge: 'badge.png'
  };
  event.waitUntil(
      self.registration.showNotification(title, options)
  );
});
```

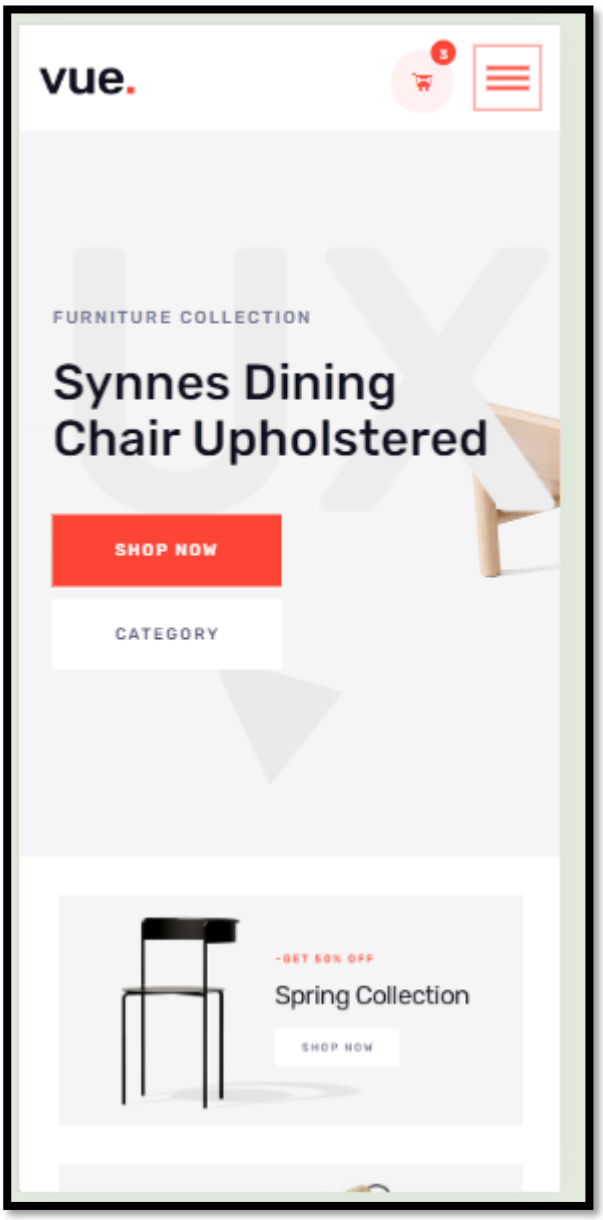
→ **Website SnapShots –**



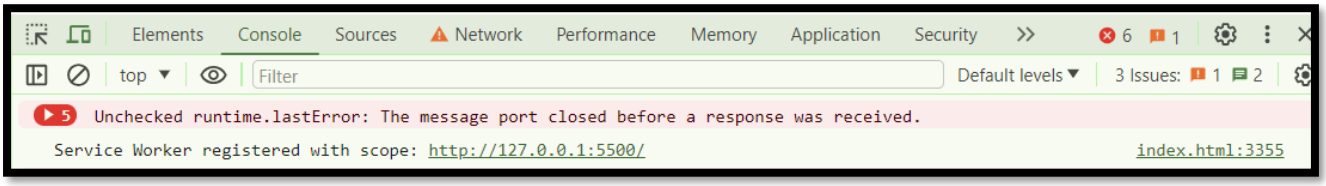


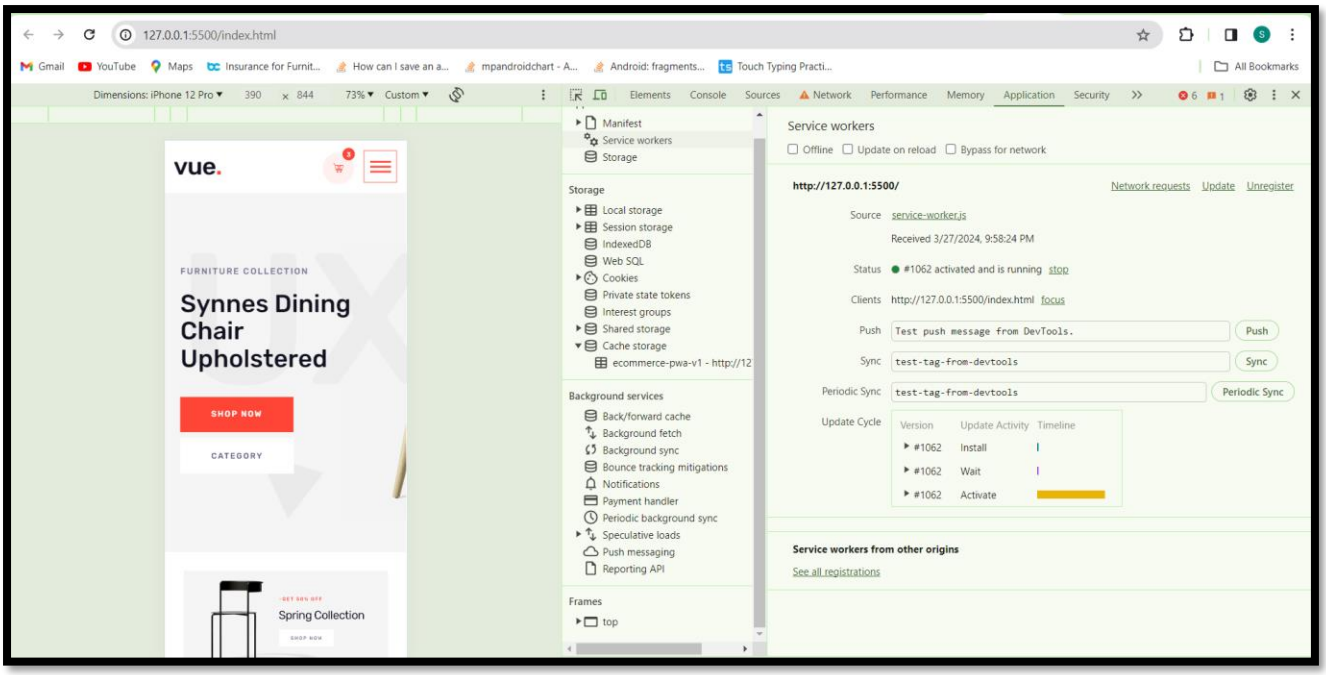


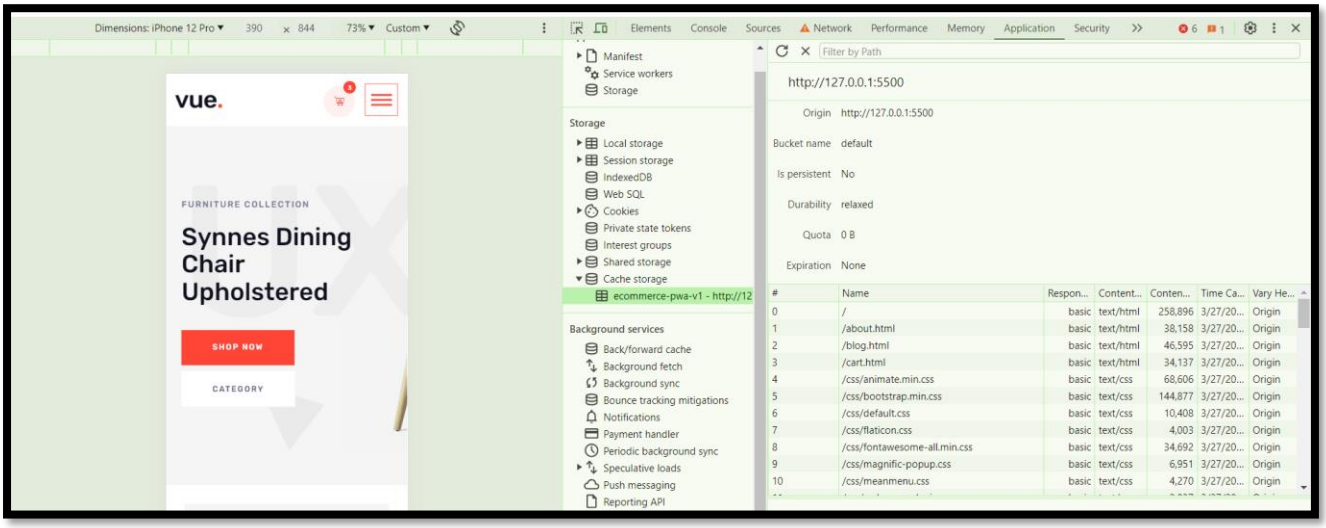


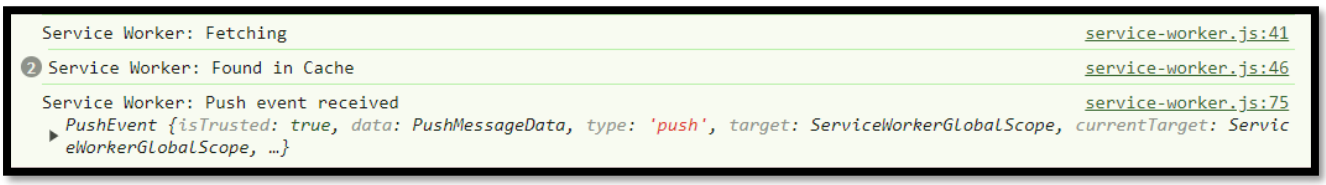


→ **Performed Fetch, Sync and Push operation –**









- **Conclusion:**

  Therefore, the integration of service worker events including fetch, sync, and push has been successfully accomplished for the E-commerce Progressive Web App (PWA). This implementation introduces advanced capabilities like offline caching, background synchronization, and push notifications. These enhancements significantly elevate user experience and ensure reliable functionality, even amidst adverse network conditions.