Practical 8

Name: Sharvari Kishor More
Class: D15B
Roll No.: 35

Aim:

To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA.

Theory:

A Service Worker is a background script that acts as a network proxy, allowing a web app to intercept network requests and serve cached content. It enhances performance and provides offline support.
In our serviceworker.js file, we implemented the following:

- **Install Event:**
  Caches important assets (index.html, CSS, images, manifest) during the installation of the service worker to make them available offline.
- **Activate Event:**
  Deletes old caches that don't match the current version to avoid storage clutter and ensure the latest files are used.
- **Fetch Event:**
  Intercepts all network requests. If the resource is in the cache, it returns the cached version; otherwise, it fetches it from the network.

This setup ensures our eCommerce PWA works smoothly even when offline and improves performance by loading resources from the cache.

Code:

Servicceworker.js

```
const CACHE_NAME = 'festival-combo-shop-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/styles.css',
  '/manifest.json',
  '/serviceworker.js',
  '/color.webp',
  '/diwalisweets.webp',
  '/holicolors.webp',
  '/christmas.webp',
];
```

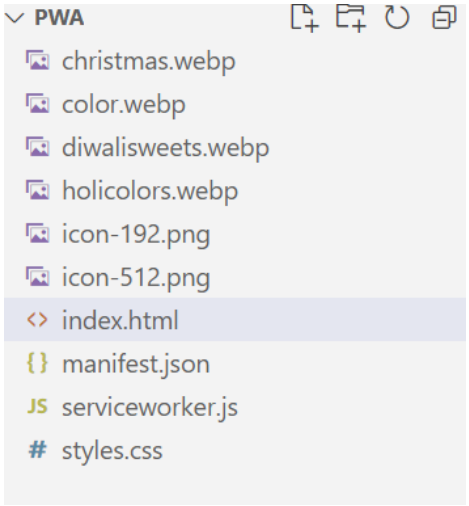
```
// Install service worker
self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('[ServiceWorker] Caching app shell');
        return cache.addAll(urlsToCache);
      })
  );
});

// Activate service worker
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(keyList => {
      return Promise.all(
        keyList.map(key => {
          if (key !== CACHE_NAME) {
            console.log('[ServiceWorker] Removing old cache', key);
            return caches.delete(key);
          }
        })
      );
    })
  );
  return self.clients.claim();
});

// Fetch content
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        // Return cached version or fetch from network
        return response || fetch(event.request);
      })
  );
});
```

Index.html

```
<script>
    if ('serviceWorker' in navigator) {
      console.log('✅ Service workers are supported in this browser.');

      window.addEventListener('load', () => {
        console.log('🌐 Window loaded. Attempting to register service worker...');

        navigator.serviceWorker.register('/serviceworker.js')
          .then(reg => {
            console.log('🎉 Service Worker registered successfully!');
            console.log('🔗 Registration scope:', reg.scope);

            if (reg.installing) {
              console.log('📦 Service Worker is installing...');
            } else if (reg.waiting) {
              console.log('⏸️ Service Worker is installed and waiting...');
            } else if (reg.active) {
              console.log('✅ Service Worker is active.');
            }

            navigator.serviceWorker.ready.then(() => {
              console.log('🚀 Service Worker is ready and controlling the page.');
            });
          })
          .catch(err => {
            console.error('❌ Service Worker registration failed:', err);
          });
      });
      navigator.serviceWorker.addEventListener('controllerchange', () => {
        console.log('🔄 Controller changed: New service worker is now controlling the page.');
      });
      navigator.serviceWorker.addEventListener('message', (event) => {
        console.log('📨 Message received from Service Worker:', event.data);
      });
    } else {
      console.warn('⚠️ Service workers are not supported in this browser.');
    }
```

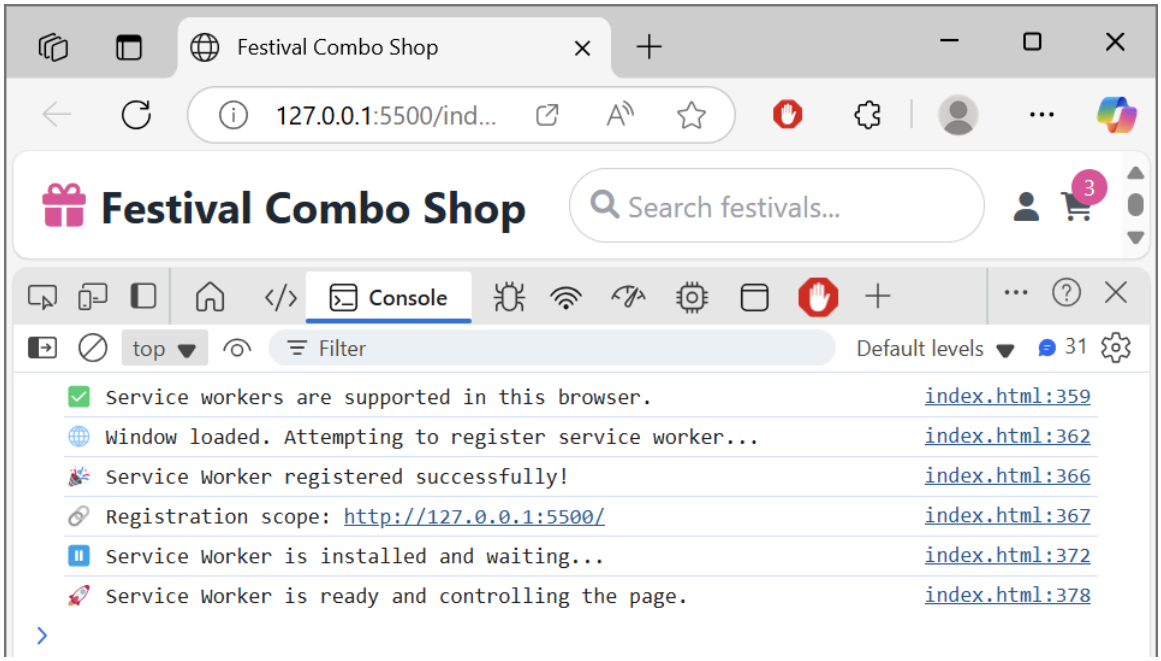
```
</script>
```

Folder Structure:

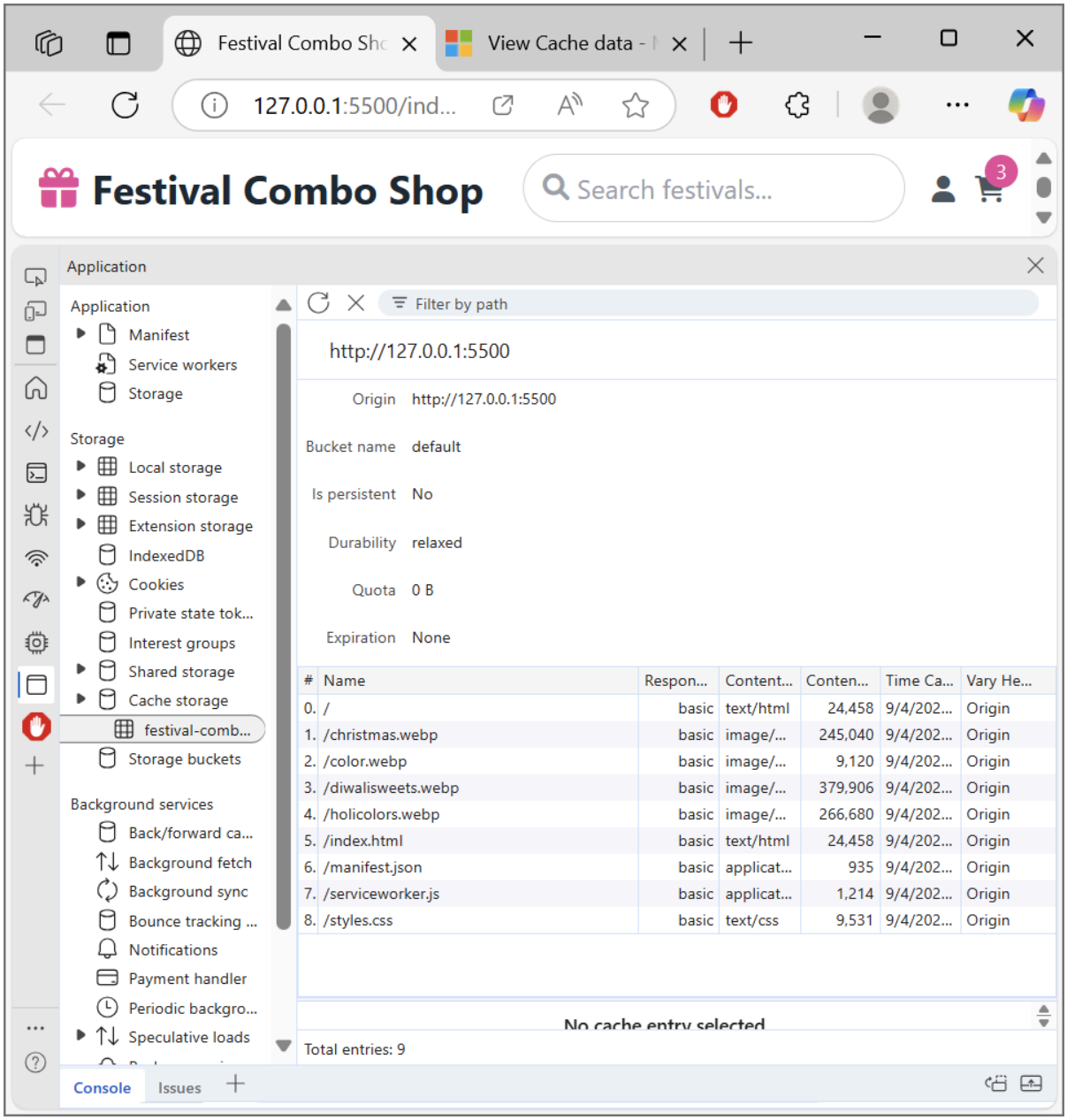


Output:

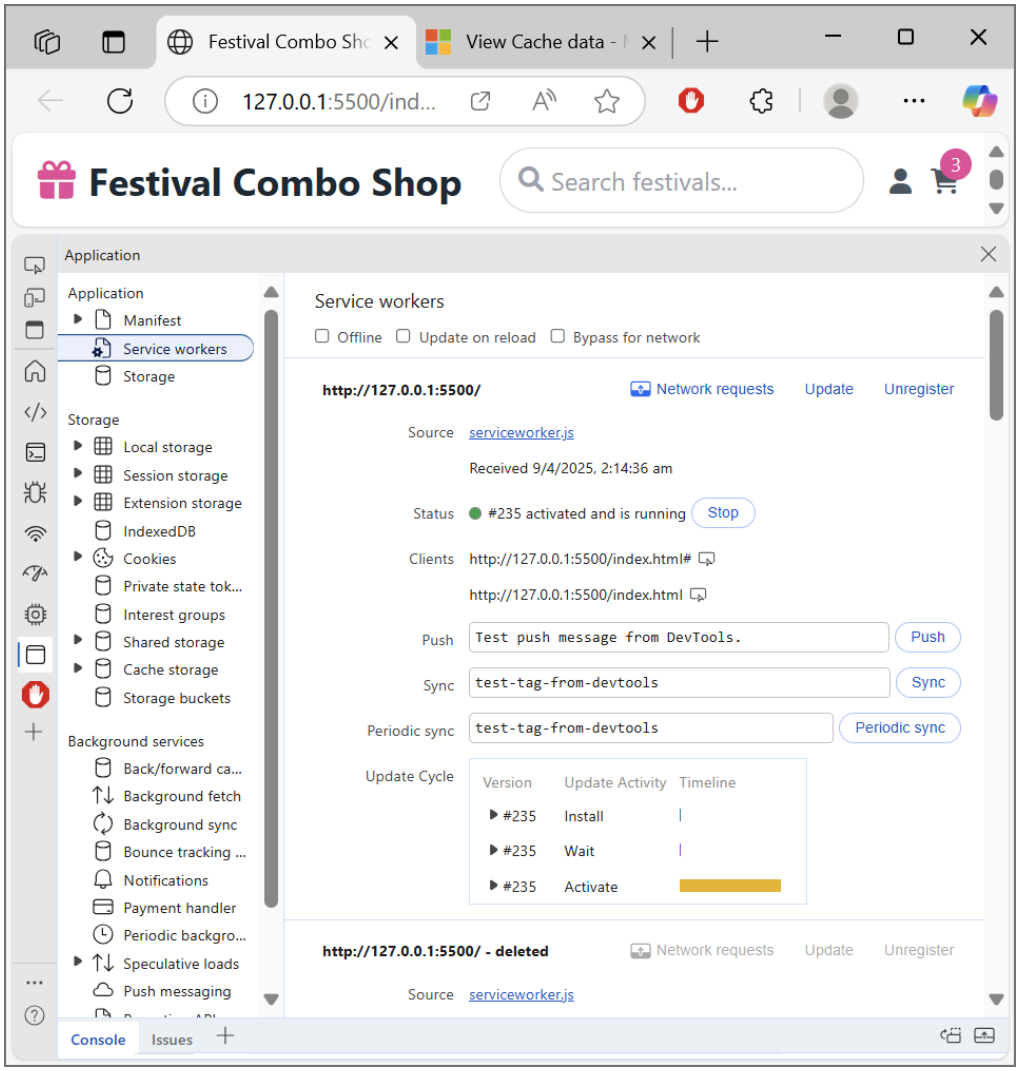
Console logs



Cache Storage



Service Worker running in the background.



Conclusion:

Through this experiment, we successfully implemented a Service Worker in our PWA, enabling offline functionality and improving load performance. It cached essential resources during installation, managed old caches on activation, and served content efficiently through fetch interception. This demonstrates how Service Workers play a crucial role in enhancing user experience in Progressive Web Apps. We faced an error when the cache storage was not being visible, so we had to delete the storage and reload the site.