Experiment 8

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.

Output:

1. **Service Worker Implementation (serviceworker.js)**

```
// Define a cache name for your app
const CACHE_NAME = 'e-commerce-pwa-v1';

// List of files to cache for offline use
const filesToCache = [
 'index.html',
 'manifest.json',
 'images/ig.jpg',
 'images/edge.jpg',
 // Add other assets your app needs
];

// Install event - Cache the essential files
self.addEventListener('install', (event) => {
 console.log('Service Worker: Installing...');
  // waitUntil() ensures the service worker won't install until
the code inside has completed
 event.waitUntil(
   caches.open(CACHE_NAME)
     .then((cache) => {
       console.log('Service Worker: Caching files');
       return cache.addAll(filesToCache);
     })
     .then(() => self.skipWaiting()) // Force waiting service
worker to become active
 );
});
```

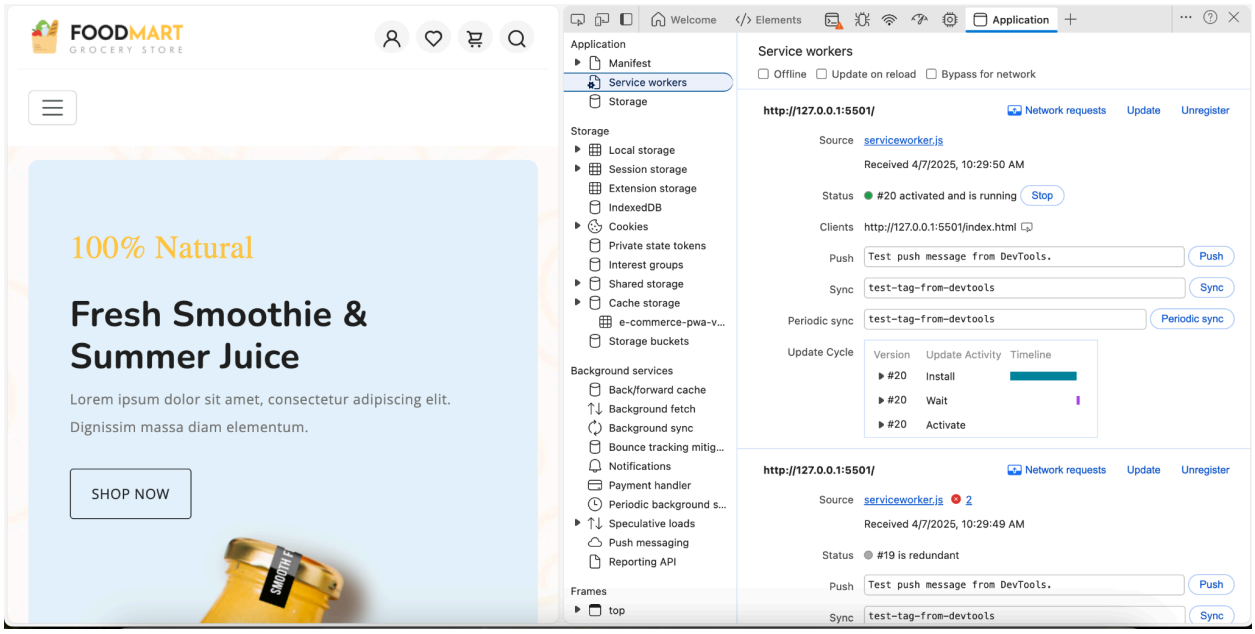
```
// Activate event - Clean up old caches
self.addEventListener('activate', (event) => {
 console.log('Service Worker: Activated');
  // Remove old caches
 event.waitUntil(
   caches.keys().then((cacheNames) => {
     return Promise.all(
       cacheNames.map((cache) => {
         if (cache !== CACHE_NAME) {
           console.log('Service Worker: Clearing old cache',
cache);
           return caches.delete(cache);
         }
       })
     );
   })
   .then(() => self.clients.claim()) // Take control of all
clients
 );
});

// Fetch event - Serve from cache first, then network
self.addEventListener('fetch', (event) => {
 console.log('Service Worker: Fetching', event.request.url);
  event.respondWith(
   // Try the cache first
   caches.match(event.request)
     .then((response) => {
       // Return cached response if found
       if (response) {
         return response;
       }

       // If not in cache, fetch from network
```

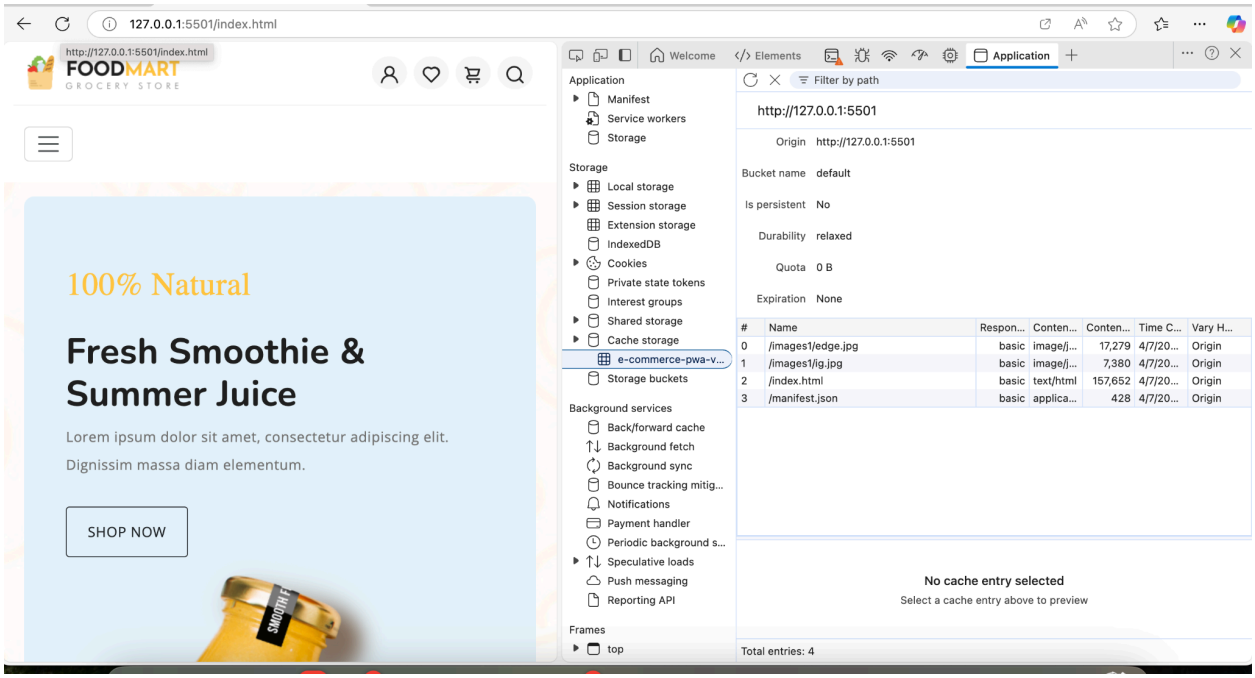
```
        return fetch(event.request)
          .then((res) => {
            // Check if valid response
            if (!res || res.status !== 200 || res.type !==
'basic') {
              return res;
            }

            // Clone the response
            let responseToCache = res.clone();

            // Add the new file to cache
            caches.open(CACHE_NAME)
              .then((cache) => {
                cache.put(event.request, responseToCache);
              });

            // Return the response
            return res;
          })
          .catch(() => {
            // If both cache and network fail, you could return
a fallback page
            if (event.request.url.indexOf('.html') > -1) {
              return caches.match('index.html');
            }
          });
      })
  );
});
```

2. Changes Made in the index.html

```
if ("serviceWorker" in navigator) {
        window.addEventListener("load", () => {
```

```
        navigator.serviceWorker
          .register("serviceworker.js")
          .then((registration) => {
            console.log(
              "ServiceWorker registration successful with
scope: ",
              registration.scope
            );
          })
          .catch((error) => {
            console.log("ServiceWorker registration failed: ",
error);
          });
      });
    }
```





Conclusion

The implementation successfully demonstrates all stages of the service worker lifecycle as outlined in the practical document. The service worker now enables offline access to the application by caching essential resources during installation and serving them when needed, providing a more reliable user experience even without an internet connection.