Experiment No: 8

Name:- Vishal Gori
Batch: A
Roll No.: 18
Division:- D15B

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:-

Progressive Web Apps (PWAs) leverage service workers to provide enhanced offline capabilities, faster loading times, and a more app-like experience for users. The service worker is a JavaScript file that runs in the background, separate from the main browser thread, and intercepts network requests made by the web application. It enables features such as caching assets, intercepting fetch requests, and managing push notifications.

To create a service worker for an E-commerce PWA, you'll follow these steps:

1. **Creating the Service Worker File:** Begin by creating a JavaScript file named service-worker.js. This file will contain the logic for caching assets and intercepting fetch requests.
2. **Registering the Service Worker:** In the main HTML file of your E-commerce PWA (e.g., index.html), add script tags to register the service worker. This typically involves using the navigator.serviceWorker.register() method inside a script tag, which registers the service worker file.

```
  <script src="./assets/js/script.js" defer>
    if ('serviceWorker' in navigator) {
      window.addEventListener('load', () => {
        navigator.serviceWorker.register('service-worker.js')
          .then(registration => {
            console.log('Service Worker registered with scope:',
registration.scope);
          })
          .catch(error => {
            console.error('Service Worker registration failed:',
error);
```

```
            });
        });
    }

</script>
```

3. **Coding the Service Worker Logic:** Inside the service-worker.js file, implement the logic for caching assets during installation, intercepting fetch requests, and activating the service worker. This typically involves adding event listeners for the install, fetch, and activate events.

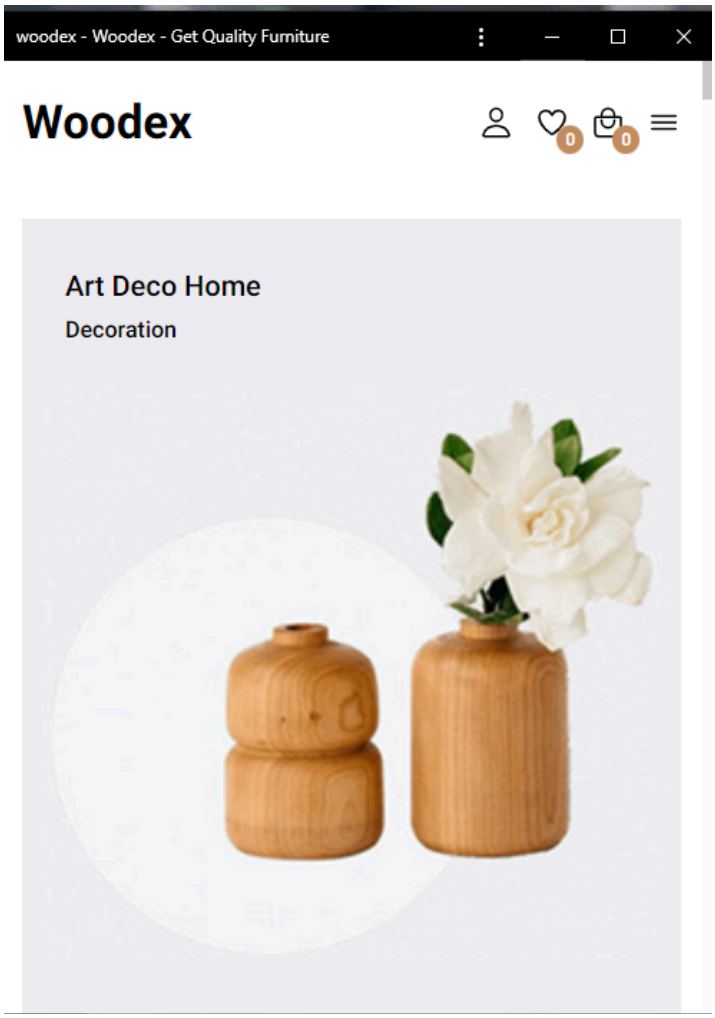
service-worker.js

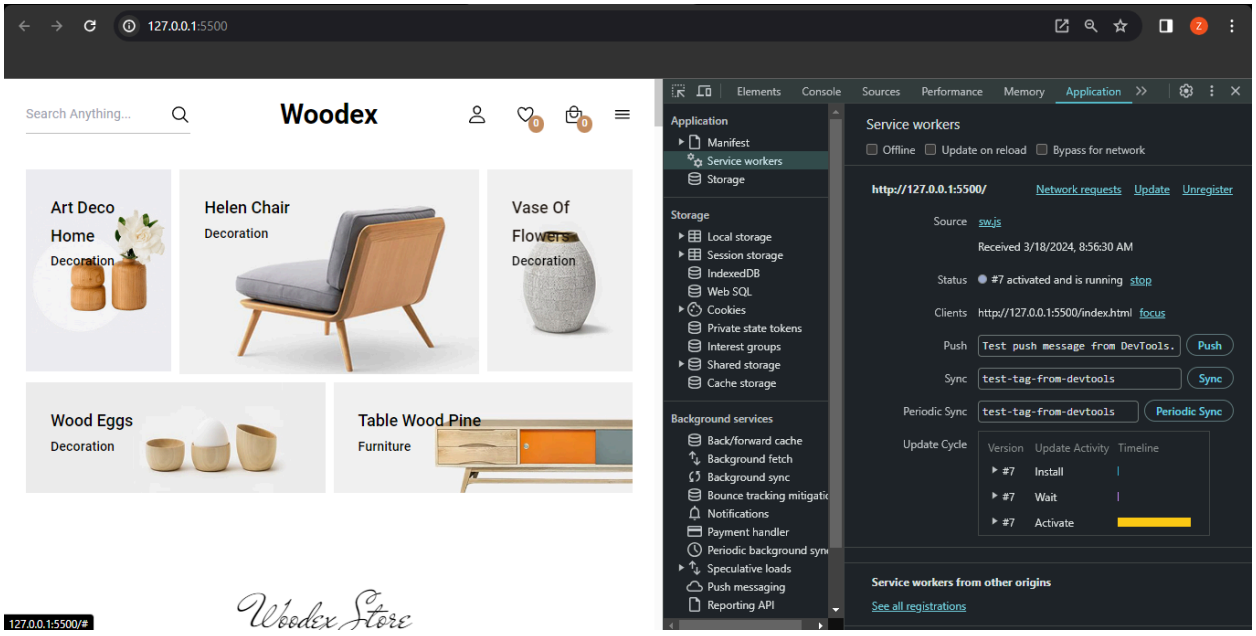
```
const CACHE_NAME = 'woodex-cache-v1';

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Opened cache');
        return cache.addAll([
          '/',
          '/index.html',
          '/styles/main.css',
          '/scripts/main.js',
          '/icons/manifest-icon-192.maskable.png',
          '/icons/manifest-icon-512.maskable.png',
          '/images/hero-product-1.jpg',
          '/images/hero-product-2.jpg',
          '/images/hero-product-3.jpg',
          '/images/hero-product-4.jpg',
          '/images/hero-product-5.jpg'
        ]);
      })
  );
});

self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
```

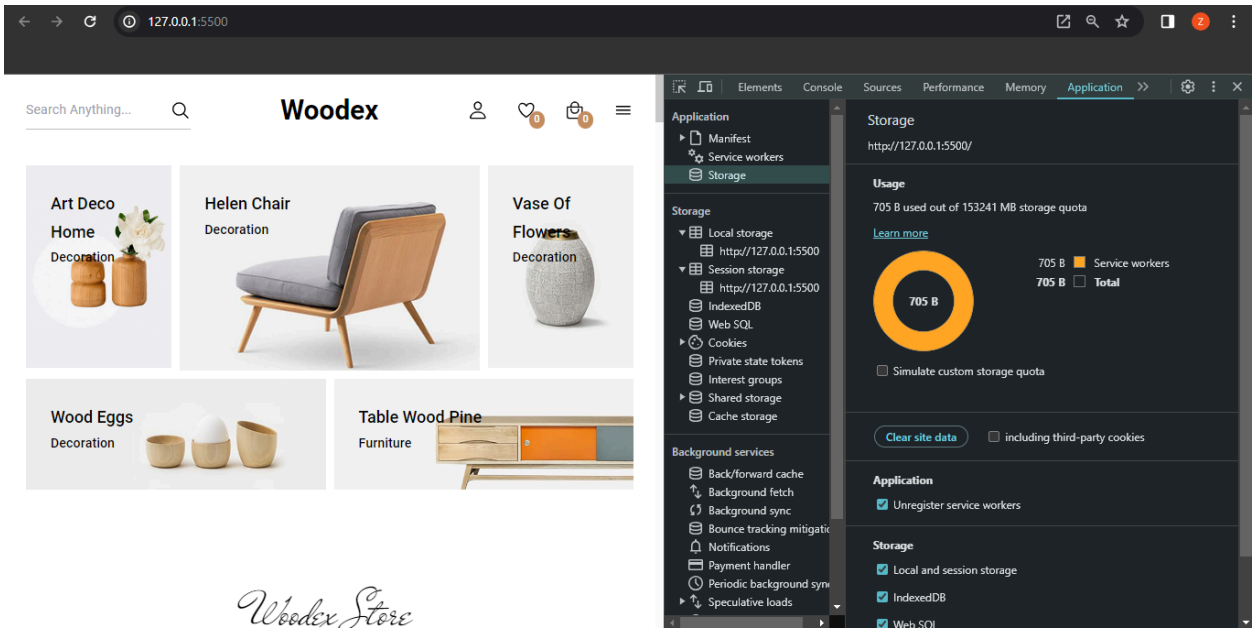
```
        if (response) {
          return response;
        }
        return fetch(event.request)
          .then(response => {
            let responseClone = response.clone();
            caches.open(CACHE_NAME)
              .then(cache => {
                cache.put(event.request, responseClone);
              });
            return response;
          })
          .catch(error => {
            console.error('Fetch failed:', error);
          });
      })
  );
});

self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.filter(cacheName => {

          return cacheName.startsWith('woodex-cache-') && cacheName !==
CACHE_NAME;
        }).map(cacheName => {
          return caches.delete(cacheName);
        })
      );
    })
  );
});
```

4. **Testing and Debugging:** Test the service worker by accessing your E-commerce PWA in a supported browser. Use the browser's developer tools to inspect network requests, check console logs, and verify that assets are being cached and served correctly.

5. **Deployment:** Once you've verified that the service worker is working as expected, deploy your E-commerce PWA to a web server to make it accessible to users.







Conclusion:- In conclusion, by coding and registering a service worker and completing the install and activation process for my E-commerce Progressive Web App (PWA), I've significantly enhanced its functionality. With offline caching capabilities, faster loading times, and improved user experience, my PWA now offers seamless access to essential features even in low or no connectivity situations.