Siddhant Bhoi D15B_07

Experiment No.8
MAD & PWA LAB

I. **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.

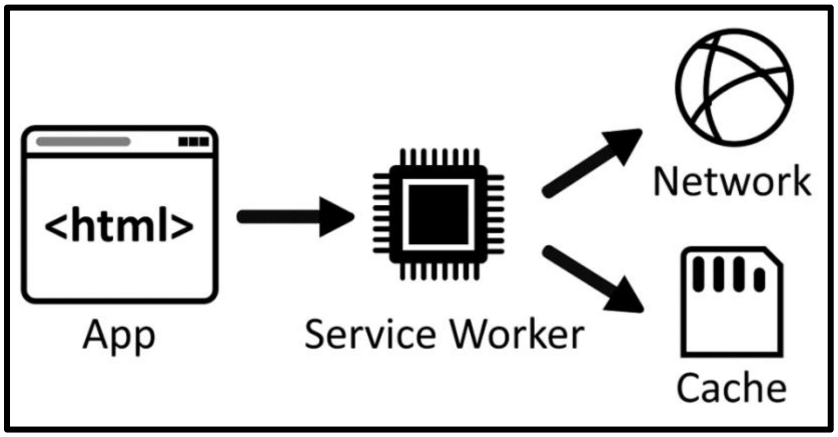
II. Theory:

Service-Worker:

Service workers essentially act as proxy servers that sit between web applications, the browser, and the network (when available). They are intended, among other things, to enable the creation of effective offline experiences, intercept network requests and take appropriate action based on whether the network is available, and update assets residing on the server. They will also allow access to push notifications and background sync APIs.

Usage of Service-Worker:

- ✓ Background data synchronization.
- ✓ Responding to resource requests from other origins.
- ✓ Receiving centralized updates to expensive-to-calculate data such as geolocation or gyroscope, so multiple pages can make use of one set of data.
- ✓ Client-side compiling and dependency management of CoffeeScript, less, CJS/AMD modules, etc. for development purposes.
- ✓ Hooks for background services.
- ✓ Custom templating based on certain URL patterns.
- ✓ Performance enhancements, for example pre-fetching resources that the user is likely to need in the near future, such as the next few pictures in a photo album.
- ✓ API mocking.



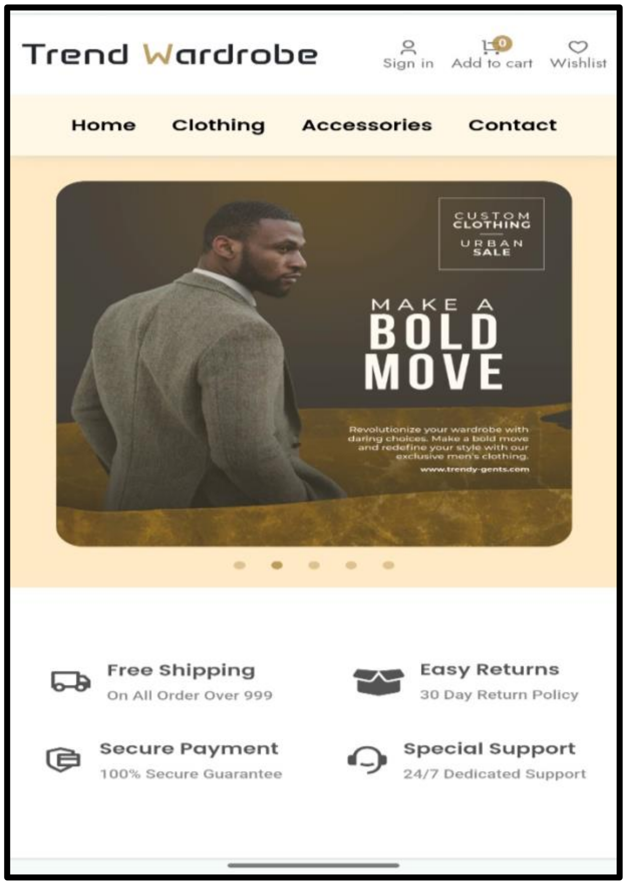
Service-Worker.js:

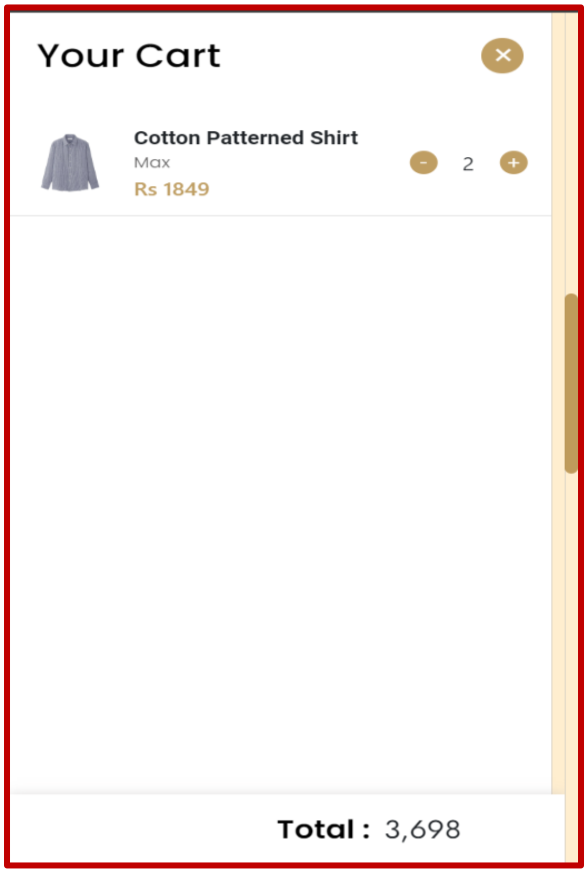
```
self.addEventListener('install', function(event) {
   console.log('Service Worker: Installed');
});
self.addEventListener('activate', function(event) {
   console.log('Service Worker: Activated');
});
self.addEventListener('fetch', function(event) {
   console.log('Service Worker: Fetching');
   event.respondWith(
      fetch(event.request)
         .then(function(response) {
            if (!response || response.status !== 200 || response.type !== 'basic') {
               return response;
            }
            var responseToCache = response.clone();
            caches.open('v1').then(function(cache) {
               cache.put(event.request, responseToCache);
            });
           return response;
         .catch(function(err) {
            console.error('Service Worker: Error fetching and caching new data:', err);
         })
   );
});
```

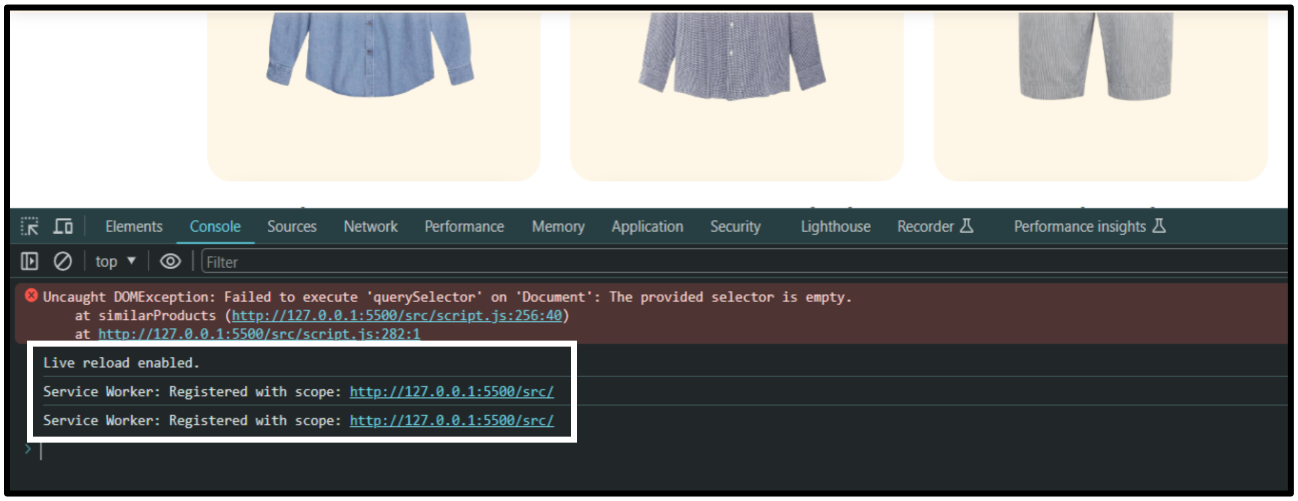
Index.html:

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

III. OUTPUT:







```
Uncaught DOMException: Failed to execute 'querySelector' on 'Document': The provided selector is empty.
    at similarProducts (http://127.0.0.1:5500/src/script.js:256:40)
    at http://127.0.0.1:5500/src/script.js:282:1
Live reload enabled.
Service Worker: Registered with scope: http://127.0.0.1:5500/src/
Service Worker: Registered with scope: http://127.0.0.1:5500/src/
```

IV. Conclusion:

Hence, successfully implemented a service worker for the E-commerce Progressive Web App (PWA) involves writing the service worker code to handle caching and network requests, registering the service worker in the main JavaScript file of the app, and completing the install and activation process.