Experiment 08

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 (Progressive Web App).

Theory:

A **Service Worker** is a background script that acts as a proxy between a web application, the browser, and the network. It enables advanced features like **offline access**, **caching**, **push notifications**, and **background sync**, which are essential for building **Progressive Web Apps (PWAs)**.

Key features of a PWA include:

- Offline support
- Improved performance through caching
- App-like behavior
- Ability to work independently of network conditions

Service workers follow a **lifecycle**, consisting of the following main events:

1. **Install**
2. **Activate**
3. **Fetch**

Implementation:

1. navigator.serviceWorker.register(): This method registers the service worker when the page loads. It checks if service workers are supported in the browser and then registers sw.js located at the root level.

2. self.addEventListener('install'): Triggered when the service worker is first installed. This is where assets are typically cached for offline use. In this case, we simply log the installation and immediately activate using self.skipWaiting().

3. **self.skipWaiting():** Forces the newly installed service worker to activate immediately rather than waiting for the old one to be terminated.

4. **self.addEventListener('activate'):** Triggered after the service worker is installed. This event is used to clean up old caches or perform updates. In our code, it's used to log activation status.

5. **self.addEventListener('fetch'):** Intercepts every network request. This is where caching or fallback responses are generally served. In our implementation, it simply logs each fetch request.

Github Execution:

https://github.com/brijeshforcollege/PWA-App

Code:

service-worker.json:

```
const CACHE_NAME = "qr-generator-cache-v1";
const urlsToCache = [
  "./",
  "./index.html",
  "./styles.css",
  "./script.js",
  "./manifest.json",
  "./images/icon-192.png",
  "./images/icon-512.png"
];

self.addEventListener("install", (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      return cache.addAll(urlsToCache);
    })
  );
});

self.addEventListener("fetch", (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request);
    })
  );
});
```

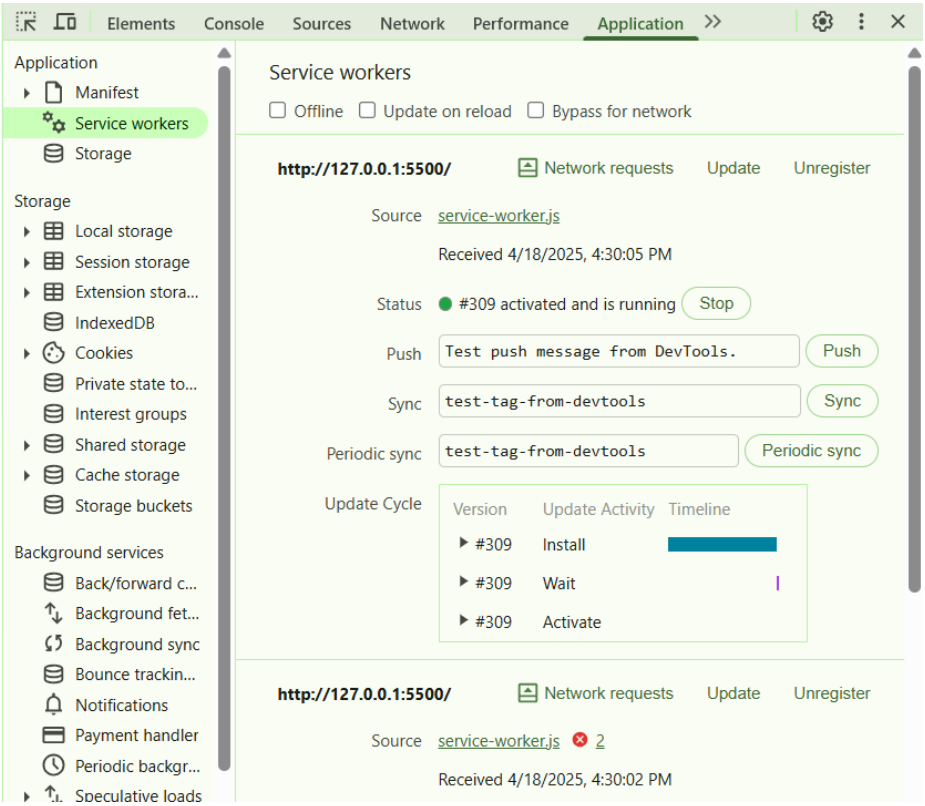
service-worker registration:

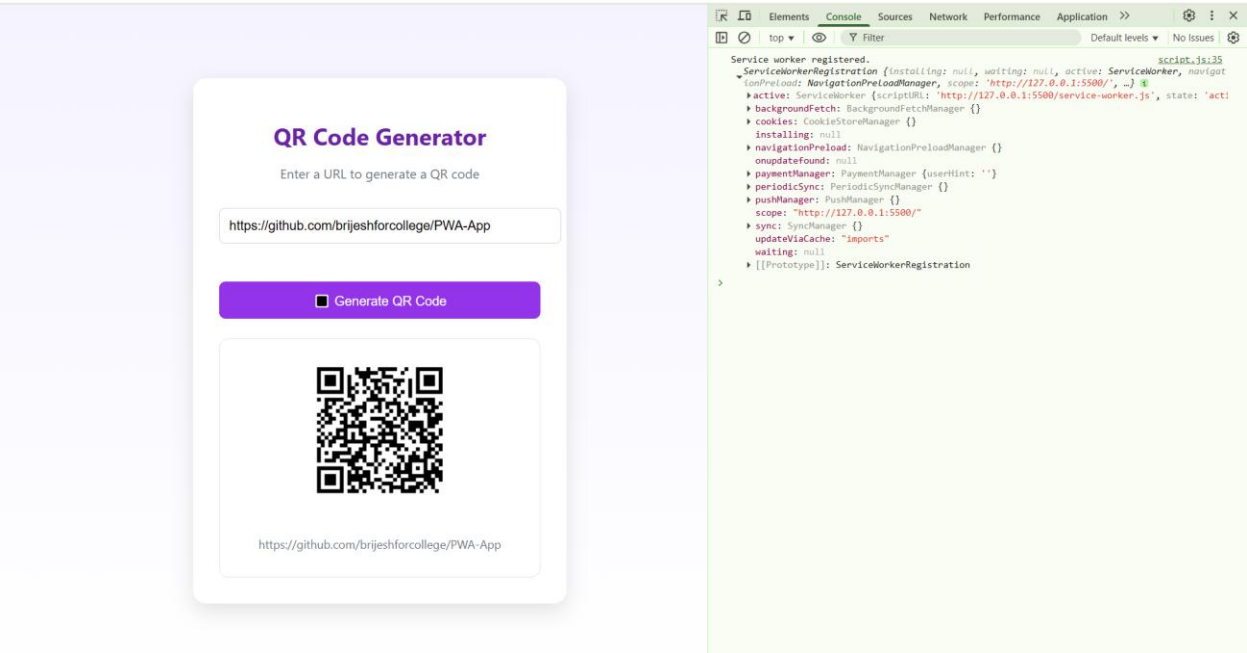
```
if ("serviceWorker" in navigator) {
   window.addEventListener("load", () => {
     navigator.serviceWorker
       .register("service-worker.js")
       .then((reg) => {
         console.log("Service worker registered.", reg);
       })
       .catch((err) => console.error("SW registration failed: ", err));
   });
 }
```

Output:

We tested the application using:

- **Chrome DevTools > Application Tab** to inspect service worker status
- **Simulated offline mode** in DevTools to check behavior when network is turned off
- Verified whether assets were served from the cache or network





Conclusion:

Through this implementation, we successfully registered and activated a service worker, gaining an understanding of key lifecycle events such as install, activate, and fetch. We observed basic Progressive Web App (PWA) behavior using browser DevTools, allowing us to see how the app functions more like a native application. This setup lays the foundation for implementing caching strategies and offline support, which are essential features in building reliable and efficient E-commerce PWAs. Overall, it enhances the user experience, particularly in environments with low or no internet connectivity.