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 client-side script that runs in the background of a web browser, separate from the main browser thread, enabling features that do not require a web page or user interaction. It acts as a programmable network proxy, allowing developers to control how network requests from their web application are handled.

Service Workers are fundamental to Progressive Web Apps (PWAs), enabling key functionalities such as:

- Offline support by caching resources
- Background data synchronization
- Push notifications

They provide the ability to intercept and respond to network requests programmatically, which can improve application performance and user experience.

Service Worker Lifecycle:

The lifecycle of a Service Worker is distinct and different from typical JavaScript running in web pages. It involves several well-defined states and events:

a) Registration: Before a Service Worker can operate, it must be registered by the web page. Registration is the process where the browser is told about the Service Worker script location, initiating its installation process.

b) Installation:

- The install event is fired when the browser downloads the Service Worker script for the first time or when it detects a change in the script.
- During installation, the Service Worker typically caches static resources (HTML, CSS, JavaScript, images) to enable offline usage.

- The installation is considered successful only if the associated promises (such as caching resources) resolve successfully.

c) Activation:

- After installation completes successfully, the Service Worker enters the activation phase.

- During activation, the Service Worker can clean up old caches from previous versions and take control of pages.

- The Service Worker only starts controlling pages after activation is complete.

d) Idle/Waiting:

- If a previous version of the Service Worker is still controlling pages, the new Service Worker waits in the "waiting" phase until all pages controlled by the old version are closed.

- The new Service Worker activates only after the old one is no longer in use unless manually forced.

e) Fetch:

- Once active, the Service Worker can intercept network requests made by the controlled pages.

- It can respond with cached resources or fetch updated resources from the network.

Detailed Explanation of Lifecycle Events:

- **Registration:**

Service Worker registration happens from within the main JavaScript thread of a web page, and typically looks like this:

```
if ('serviceWorker' in navigator) { navigator.serviceWorker.register('/sw.js')

.then(registration => {

console.log('Service Worker registered with scope:', registration.scope);
```

```
})
.catch(error => {
console.error('Service Worker registration failed:', error);
});
}
```

- navigator.serviceWorker.register() tells the browser where the Service Worker file is located.
- The registration returns a promise that resolves if registration succeeds or rejects on failure.
- **Installation:**

Once registered, the browser attempts to install the Service Worker by firing the install event:

```
self.addEventListener('install', event => { event.waitUntil(
caches.open('static-cache-v1').then(cache => { return cache.addAll([
'/',
'/index.html', '/styles.css', '/script.js'
]);
})
);
});
```

- The install event allows the Service Worker to prepare resources (usually by caching).

- event.waitUntil() ensures that the Service Worker does not install until the caching completes successfully.

- If the installation fails (e.g., cache promise rejects), the Service Worker will not install, and the browser may retry later.

- **Activation:**

After installation, the Service Worker moves to the activation phase:

```
self.addEventListener('activate', event => { event.waitUntil( caches.keys().then(cacheNames => {

return Promise.all(

cacheNames.filter(name => name !== 'static-cache-v1')

.map(name => caches.delete(name))

);

})

);

});
```

- The activate event is an opportunity to remove outdated caches or perform other maintenance.

- The Service Worker only starts controlling clients after this event completes.

- Fetch Event (Interception of Network Requests)

After activation, the Service Worker can intercept network requests using the fetch event:

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

return response || fetch(event.request);

})
```

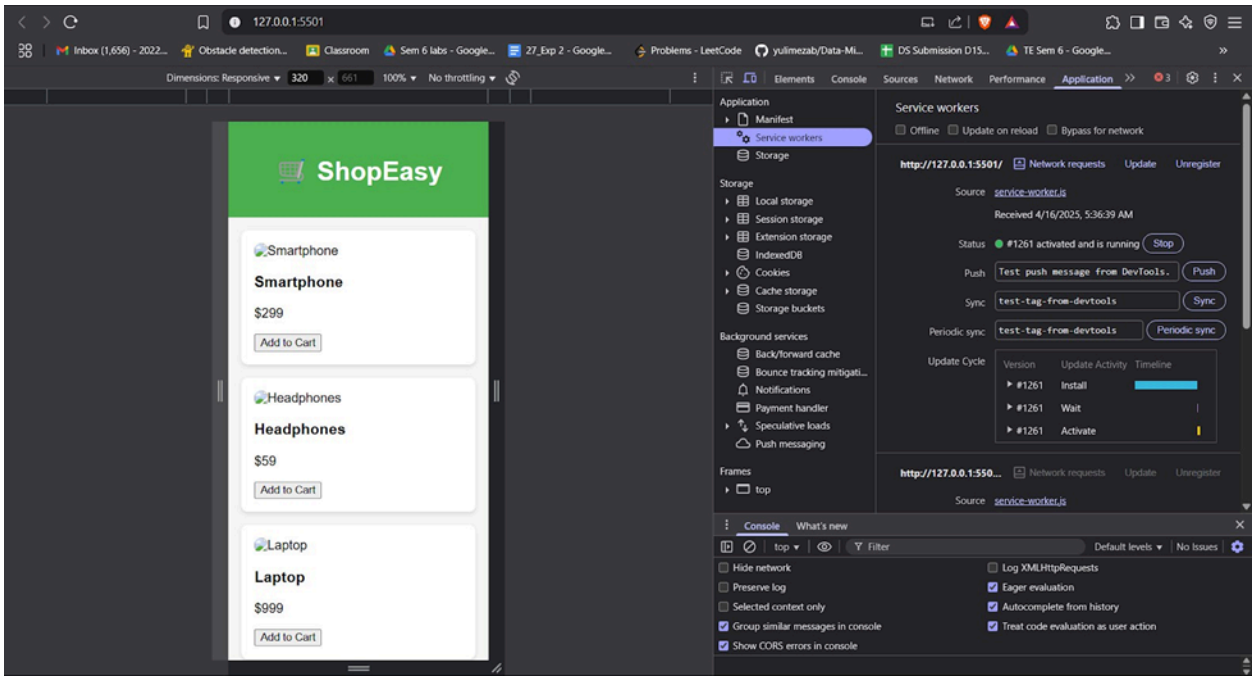
);

});

- The Service Worker first attempts to serve the requested resource from the cache.
- If not found, it fetches the resource from the network.
- This allows for offline capabilities and improved performance.

Code: https://github.com/Kingmaker-2/PWA-8.git

Output:



Developer tools showing the installation and working of Service Worker

Conclusion: Service Workers are a powerful feature of modern web browsers that enable developers to create reliable, fast, and engaging web applications by controlling network requests and managing resource caching. Understanding the Service Worker lifecycle—comprising registration, installation, activation, and fetch interception—is crucial for effective implementation. Through installation, essential resources can be cached to allow offline functionality, while activation ensures old caches are cleaned and the new Service Worker gains control. By intercepting

network requests, Service Workers provide significant performance improvements and resilience against unreliable network conditions. Mastery of Service Workers forms the foundation for building Progressive Web Apps that deliver a native-app-like user experience on the web.