Name - Gaurang A Raorane Div - D15A
Roll no - 49 Batch - C

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.

Theory -
Service Worker is a script that operates independently in the background of a browser, acting as a programmable network proxy. It enables developers to control how network requests from a web page are handled. It's crucial to note that Service Workers only function over HTTPS due to security concerns, as they have the capability to intercept network requests and modify responses.

Here's a summary of what you can do and can't do with Service Workers:

What You Can Do:

1. Dominate Network Traffic: You can manage all network traffic of the page and manipulate requests and responses as needed. This includes altering responses for different types of requests.

2. Cache Content: Service Workers allow caching of request/response pairs using the Cache API. This enables offline access to cached content, improving performance and user experience.

3. Manage Push Notifications: You can handle push notifications with Service Workers, allowing you to display messages to users even when the internet connection is broken.

4. Background Sync: Service Workers facilitate background processes, enabling tasks to continue despite interruptions in internet connectivity.

What You Can't Do:

1. Access the Window: Service Workers cannot access the window or manipulate DOM elements directly. Communication with the window is possible through postMessage.

2. Work on Port 80: Service Workers only function on HTTPS, but during development, they can operate on localhost.

Registration:

To install a Service Worker, it must be registered in the main JavaScript code. This registration informs the browser where the Service Worker is located and initiates the installation process in the background.

Installation:
Once registered, the browser attempts to install the Service Worker if it's considered new or updated. This triggers an install event, during which tasks such as precaching parts of the web app can be performed to enhance subsequent loading times.

Overall, Service Workers offer powerful capabilities for enhancing web applications, particularly in terms of offline functionality, network traffic management, and push notifications. However, developers must be mindful of security considerations and the limitations imposed by the Service Worker environment.

Code -
Serviceworker.js

```
const CACHE_NAME = 'my-ecommerce-app-cache-v1';
const urlsToCache = [
  '/',
  'index.html',
  'assets/js/script.js',
  'assets/css/style.css',
  'serviceworker.js',
  'manifest.json',
  'offline.html'
  // Add more files to cache as needed
];

self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(function(cache) {
        console.log('Opened cache');
        return cache.addAll(urlsToCache)
          .catch(function(error) {
            console.error('Cache.addAll error:', error);
          });
      })
  );
});

self.addEventListener('activate', function(event) {
```

```
  // Perform activation steps
  event.waitUntil(
    caches.keys().then(function(cacheNames) {
      return Promise.all(
        cacheNames.map(function(cacheName) {
          if (cacheName !== CACHE_NAME) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
self.addEventListener('activate', async () => {
  if (Notification.permission !== 'granted') {
    try {
      const permission = await Notification.requestPermission();
      if (permission === 'granted') {
        console.log('Notification permission granted.');
      } else {
        console.warn('Notification permission denied.');
      }
    } catch (error) {
      console.error('Failed to request notification permission:', error);
    }
  }
});
```

manifest.js

```
{
    "name":"PWA Tutorial",
    "short_name":"PWA",
    "start_url":"index.html",
    "display":"standalone",
    "background_color":"#5900b3",
    "theme_color":"black",
    "scope": ".",
    "description":"This is a PWA tutorial.",
    "icons":[
    {
    "src":"icon-192x192.png",
    "sizes":"192x192",
    "type":"image/png",
    "purpose":"any maskable"
```

```
    },
    {
    "src":"icon-512x512.png",
    "sizes":"512x512",
    "type":"image/png",
    "purpose":"any maskable"
    }
]
}
```

Offline.html

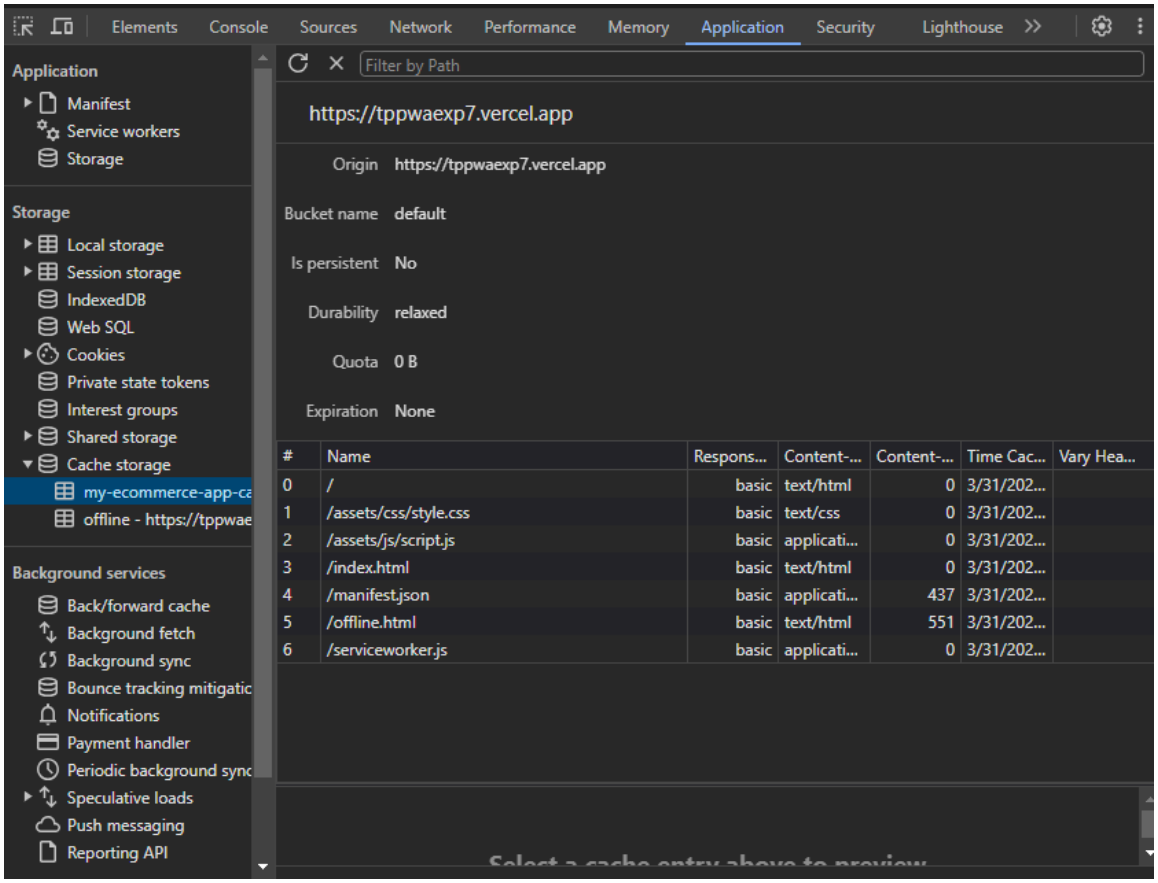
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Offline</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f0f0f0;
      color: #333;
      text-align: center;
      padding: 50px;
    }
    h1 {
      margin-bottom: 20px;
    }
  </style>
</head>
<body>
  <h1>Oops! You're offline.</h1>
  <p>It seems like you're currently offline. Please check your internet connection and try again
later.</p>
</body>
</html>
```

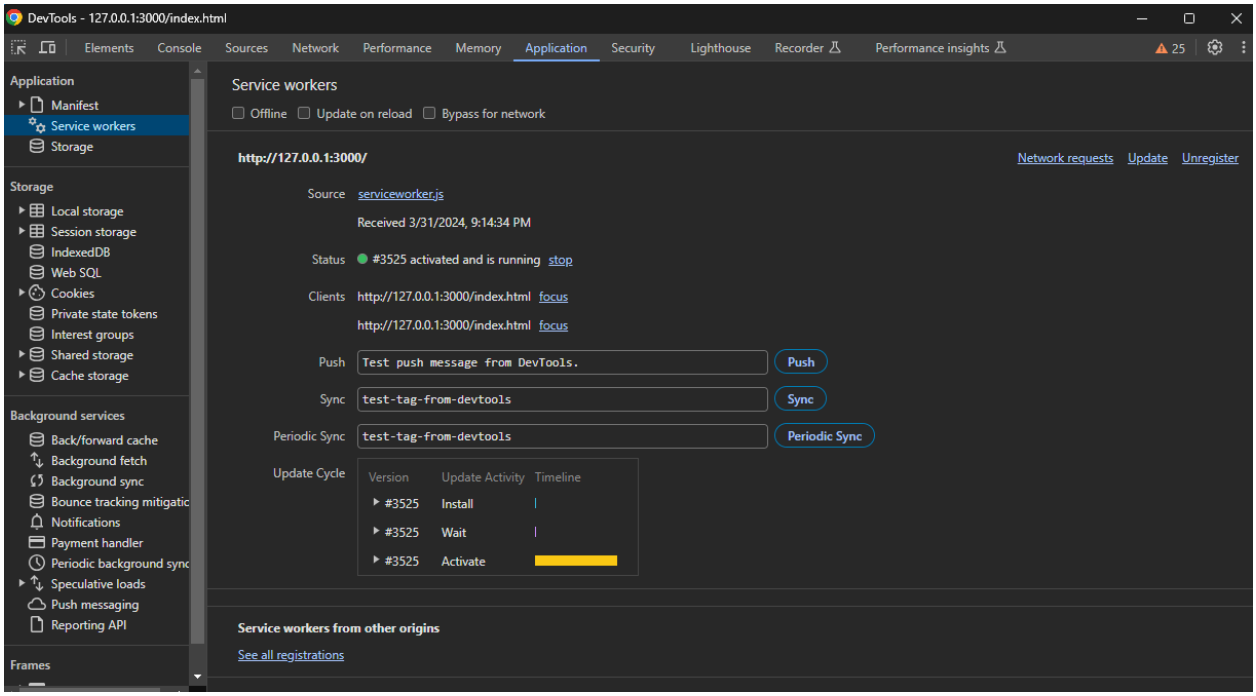
script.js-

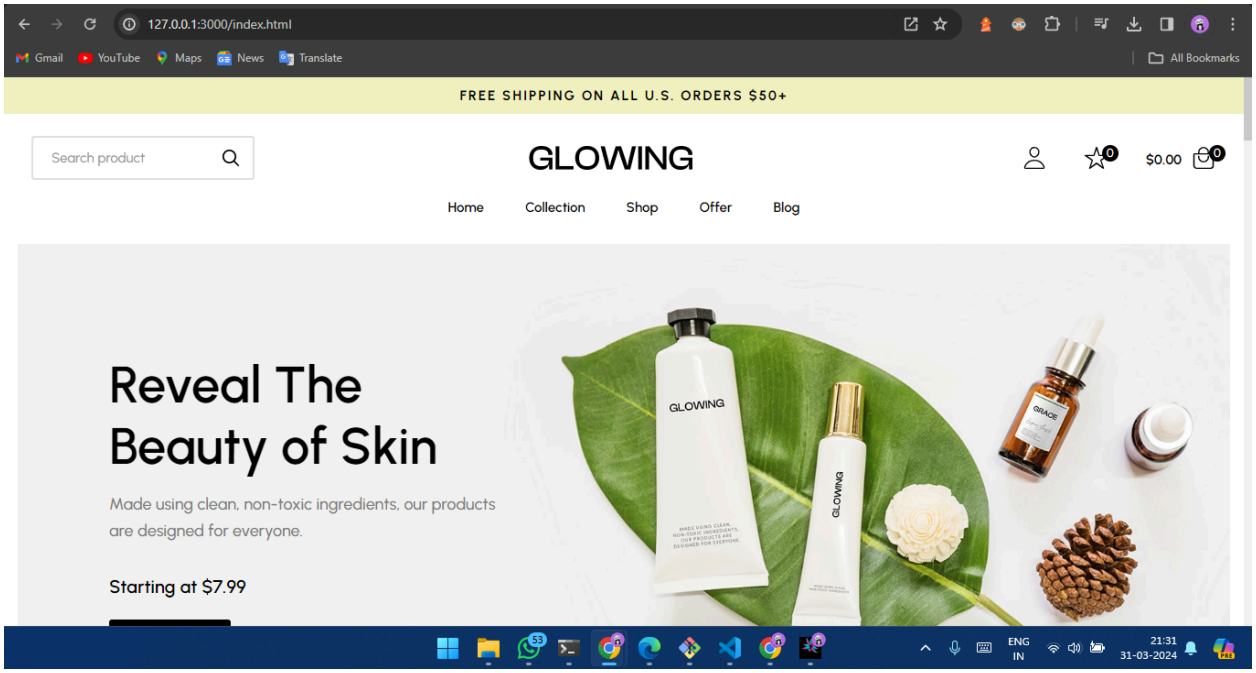
```
'use strict';

const addEventOnElem = function (elem, type, callback) {
  if (elem.length > 1) {
    for (let i = 0; i < elem.length; i++) {
      elem[i].addEventListener(type, callback);
```

```
    }
  } else {
    elem.addEventListener(type, callback);
  }
}
const navTogglers = document.querySelectorAll("[data-nav-toggler]");
const navbar = document.querySelector("[data-navbar]");
const navbarLinks = document.querySelectorAll("[data-nav-link]");
const overlay = document.querySelector("[data-overlay]");

const toggleNavbar = function () {
  navbar.classList.toggle("active");
  overlay.classList.toggle("active");
}

addEventOnElem(navTogglers, "click", toggleNavbar);

const closeNavbar = function () {
  navbar.classList.remove("active");
  overlay.classList.remove("active");
}

addEventOnElem(navbarLinks, "click", closeNavbar);
const header = document.querySelector("[data-header]");
const backTopBtn = document.querySelector("[data-back-top-btn]");

const headerActive = function () {
  if (window.scrollY > 150) {
    header.classList.add("active");
    backTopBtn.classList.add("active");
  } else {
    header.classList.remove("active");
    backTopBtn.classList.remove("active");
  }
}

addEventOnElem(window, "scroll", headerActive);

let lastScrolledPos = 0;

const headerSticky = function () {
  if (lastScrolledPos >= window.scrollY) {
    header.classList.remove("header-hide");
  } else {
```

```
    header.classList.add("header-hide");
  }

  lastScrolledPos = window.scrollY;
}
addEventOnElem(window, "scroll", headerSticky);
const sections = document.querySelectorAll("[data-section]");
const scrollReveal = function () {
  for (let i = 0; i < sections.length; i++) {
    if (sections[i].getBoundingClientRect().top < window.innerHeight / 2) {
      sections[i].classList.add("active");
    }
  }
}
scrollReveal();
addEventOnElem(window, "scroll", scrollReveal);
```







Conclusion -

In this experiment, we have registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.