MAD & PWA Lab

Journal

Experiment No.	07
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the PWA.
Roll No.	20
Name	Madhura Jangale
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Aim: To code and register a service worker, and complete the install and activation process for a new service worker for the PWA.

Theory:

Service Worker

Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API.

Things to note about Service Worker:

- A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
- The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.

What can we do with Service Workers?

- You can dominate **Network Traffic**
  You can manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true response too.

- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.

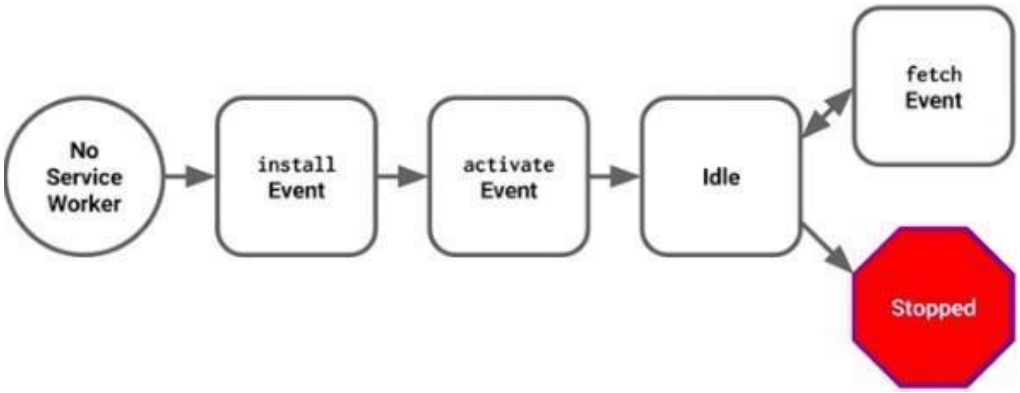
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can’t we do with Service Workers?

- You can’t access the **Window**
  You can’t access the window, therefore, You can’t manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.

- You can’t work it on **80 Port**
  Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle



A service worker goes through three steps in its life cycle:

- Registration
- Installation
- Activation

Registration
To install a service worker, you need to register it in your main JavaScript code

```
if ('serviceWorker' in navigator) {
navigator.serviceWorker.register('/service-worker.js')
.then(function(registration) {
console.log('Registration successful, scope is:', registration.scope);
})
.catch(function(error) {
console.log('Service worker registration failed, error:', error);
```

```
});
}
```

This code starts by checking for browser support by examining **navigator.serviceWorker**. The service worker is then registered with navigator.serviceWorker.register, which returns a promise that resolves when the service worker has been successfully registered. The scope of the service worker is then logged with registration.scope. If the service worker is already installed, navigator.serviceWorker.register returns the registration object of the currently active service worker.

The scope of the service worker determines which files the service worker controls, in other words, from which path the service worker will intercept requests. The default scope is the location of the service worker file, and extends to all directories below. So if service-worker.js is located in the root directory, the service worker will control requests from all files at this domain.

```
// Service Worker Script

const CACHE_NAME = 'blogbreeze-v1'; // Cache name to identify the version of cached content
const urlsToCache = [
'/', // Home page
'index.html', // Main HTML file
'ani.html', // Any additional pages you want to cache
'/src/assets/react.svg', // App icon
'/public/icon.png', // Favicon
'/src/main.jsx', // Main JS file for app functionality
//  'https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css' // FontAwesome for icons
];

// Install Service Worker
self.addEventListener('install', (event) => {
console.log('Service Worker: Installed');
event.waitUntil(
caches.open(CACHE_NAME)
.then((cache) => {
return cache.addAll(urlsToCache);
})
);
});

// Activate Service Worker
self.addEventListener('activate', (event) => {
console.log('Service Worker: Activated');
```

```
// Remove old caches if there are any
event.waitUntil( caches.keys().then((cacheNames)
=> {
return Promise.all( cacheNames.map((cacheName)
=> {
if (cacheName !== CACHE_NAME) { return
caches.delete(cacheName);
}
})
);
})
);
});

// Fetch event: Intercept network requests and serve cached content
self.addEventListener('fetch', (event) => {
console.log('Service Worker: Fetching', event.request.url);
event.respondWith(
caches.match(event.request)


.then((cachedResponse) => {
// Return cached content if found, otherwise fetch from network return
cachedResponse || fetch(event.request);
})
);
})
```

Code:

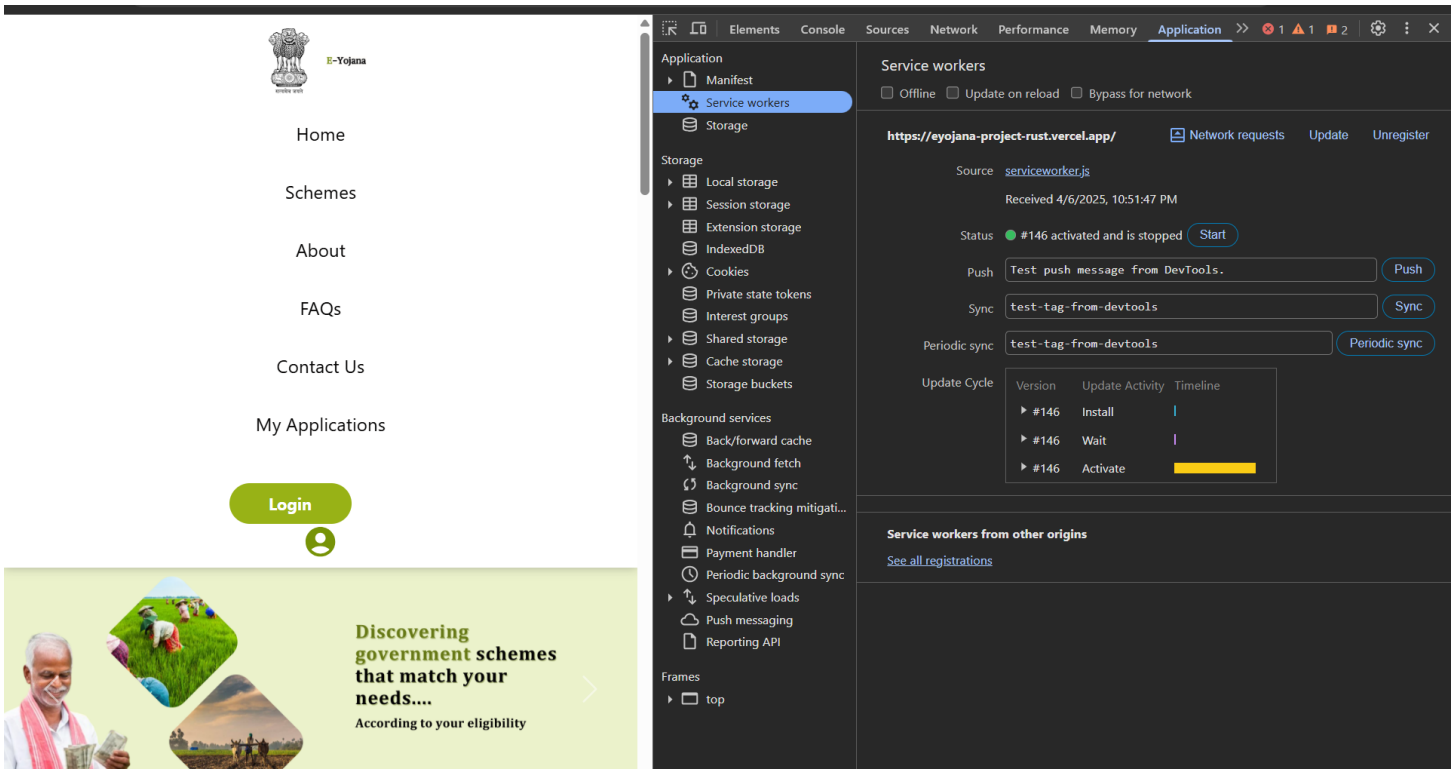
We will need to create another file for the PWA, that is, serviceworker.js in the same directory. This file handles the configuration of a service worker that will manage the working of the application.

```
var staticCacheName = "eyojana";

self.addEventListener("install", function (e) {
e.waitUntil(
    caches.open(staticCacheName).then(function (cache) {
    return cache.addAll(["/"]);
    })
);
});
self.addEventListener("fetch", function (event) {
console.log(event.request.url);

event.respondWith(
```

```
    caches.match(event.request).then(function (response) {
    return response || fetch(event.request);
    })
);
});
```



Conclusion : Thus we have learnt to code and register a service worker, and complete the install and activation process for a new service.