Experiment No: 08

Aim: To implement service worker registration and complete the install and activation process for a new service worker for an E-commerce Progressive Web App (PWA).

Theory:

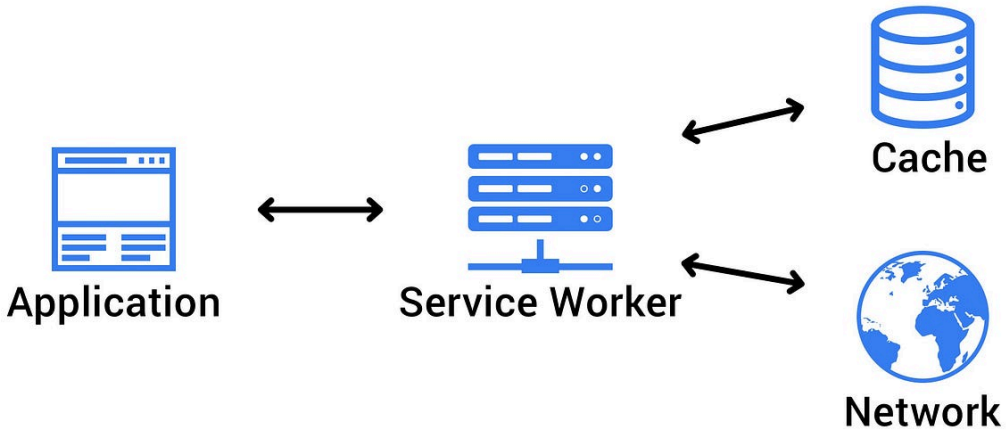
Service Worker:
A service worker is a script that operates in the background of a web browser independently of user interaction. It acts as a programmable network proxy, allowing you to control how network requests from your web page are handled. Service workers enable various functionalities such as tracking network traffic, managing push notifications, and developing "offline-first" web applications using the Cache API.

Key Points about Service Workers:
- Service workers operate as network proxies, allowing you to manipulate network requests and responses.
- They only run over HTTPS to prevent security vulnerabilities.
- Service workers have a life cycle consisting of registration, installation, and activation phases.
- They can cache resources for offline use, manage push notifications, and perform background sync operations.

What We Can Do with Service Workers:
- Control Network Traffic: Manipulate network requests and responses, cache resources, and manage offline content.
- Cache Resources: Store resources locally using the Cache API for offline access.
- Manage Push Notifications: Handle push notifications and display messages to users.
- Background Sync: Continue processes even when the internet connection is interrupted.

What We Can't Do with Service Workers:
- Access the Window: Service workers cannot access the DOM directly but can communicate with the window using postMessage.
- Operate on Port 80: Service workers require HTTPS and cannot work on Port 80.

Service Worker Cycle:
1. Registration: Register the service worker in your main JavaScript code to instruct the browser where to find it. This initiates the installation process.
2. Installation: During installation, the service worker caches essential resources for offline access and handles any precaching tasks.
3. Activation: After installation, the service worker enters the activation phase, where it takes control of pages within its scope. It cleans up outdated caches and prepares to handle network requests.

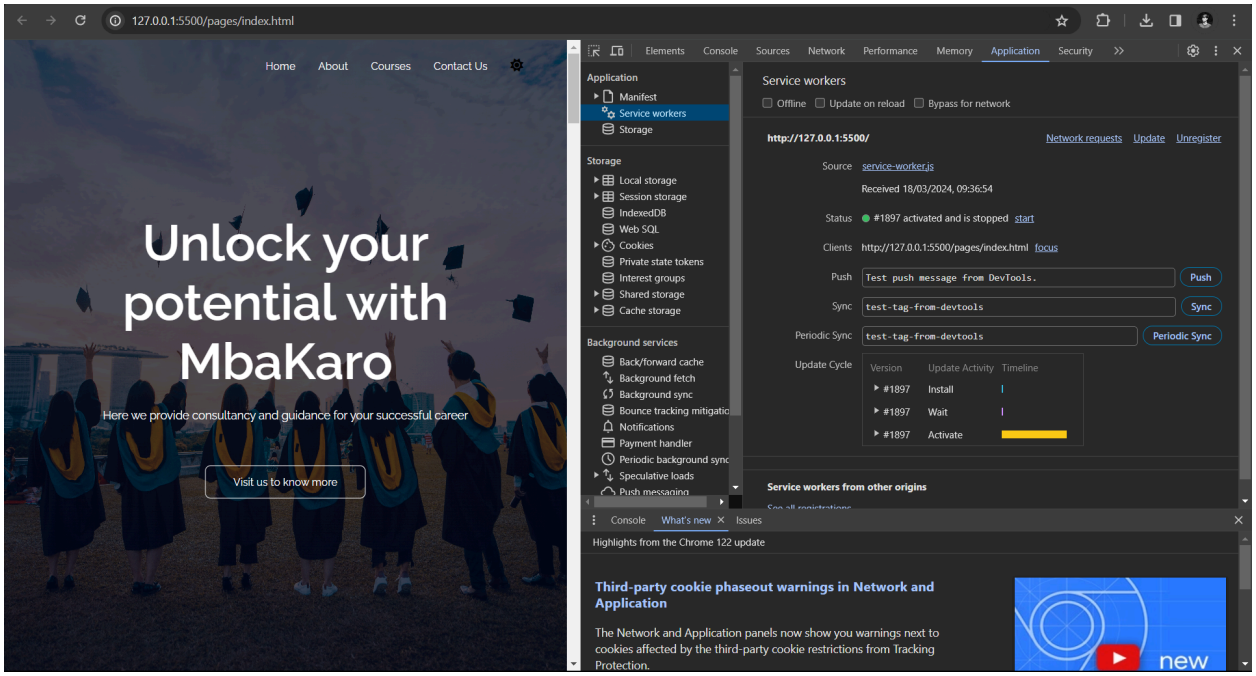


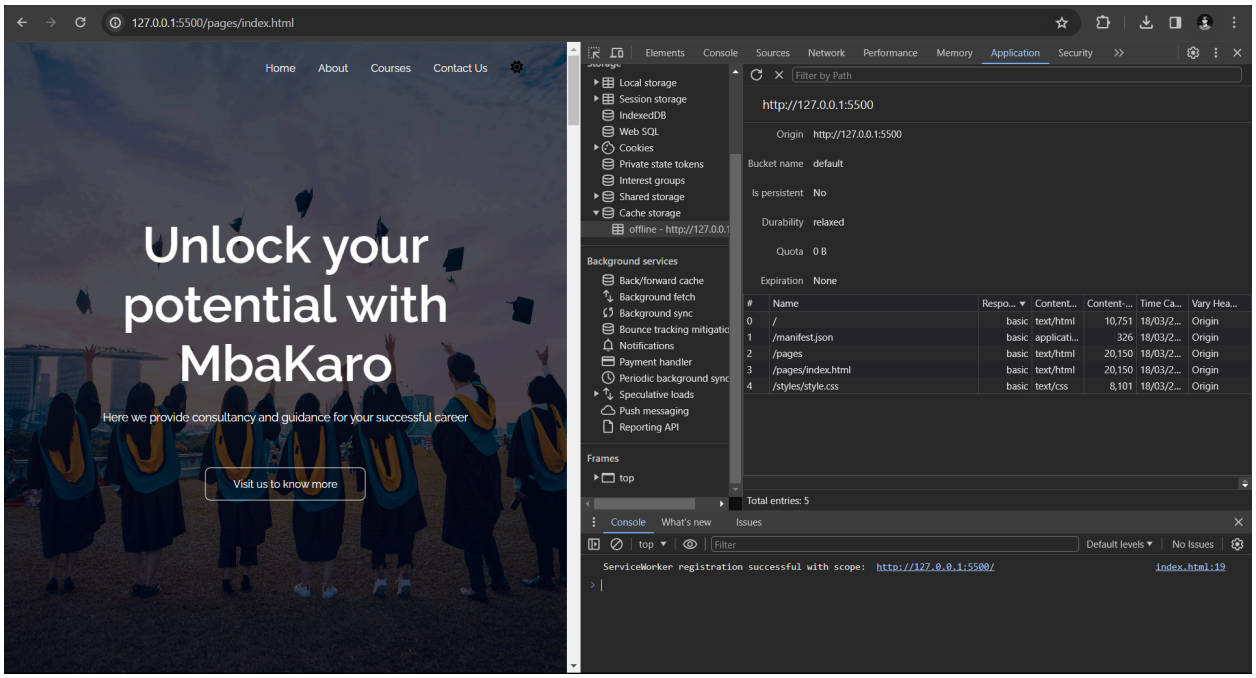
Code Example:

service-worker.js:
javascript

```
self.addEventListener("install", function (event) {
    event.waitUntil(preLoad());
});

var filesToCache = [
    '/',
    '/menu',
    '/contactUs',
    '/offline.html',
];

var preLoad = function () {
    return caches.open("offline").then(function (cache) {
        return cache.addAll(filesToCache);
    });
};

self.addEventListener("fetch", function (event) {
    event.respondWith(checkResponse(event.request).catch(function () {
        return returnFromCache(event.request);
    }));
    event.waitUntil(addToCache(event.request));
```

```
});

var checkResponse = function (request) {
    return new Promise(function (fulfill, reject) {
        fetch(request).then(function (response) {
            if (response.status !== 404) {
                fulfill(response);
            } else {
                reject();
            }
        }, reject);
    });
};

var addToCache = function (request) {
    return caches.open("offline").then(function (cache) {
        return fetch(request).then(function (response) {
            return cache.put(request, response);
        });
    });
};

var returnFromCache = function (request) {
    return caches.open("offline").then(function (cache) {
        return cache.match(request).then(function (matching) {
            if (!matching || matching.status == 404) {
                return cache.match("offline.html");
            } else {
                return matching;
            }
        });
    });
};
```





Conclusion:

Service workers play a crucial role in enhancing web applications by enabling features such as offline access, push notifications, and background sync. Understanding their life cycle and capabilities is essential for developing robust and efficient Progressive Web Apps.