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
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. Registration tells the browser where your service worker is located, and to start installing it in the background. Let's look at an example:

Main.js

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/ser
  vice-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.

You can also set an arbitrary scope by passing in an additional parameter when

registering.

For example: main.js

```
navigator.serviceWorker.register('/service-
  worker.js', { scope: '/app/'
});
```

In this case we are setting the scope of the service worker to /app/, which means the service worker will control requests from pages like /app/, /app/lower/ and /app/lower/lower, but not from pages like /app or /, which are higher.

If you want the service worker to control higher pages e.g. /app (without the trailing slash) you can indeed change the scope option, but you'll also need to set the Service-Worker-Allowed HTTP Header in your server config for the request serving the service worker script.

Main.js

```
navigator.serviceWorker.register('/app/servi
  ce-worker.js', { scope: '/app'
});
```

Installation

Once the browser registers a service worker, installation can be attempted. This occurs if the service worker is considered to be new by the browser, either because the site currently doesn't have a registered service worker, or because there is a byte difference between the new service worker and the previously installed one.

A service worker installation triggers an install event in the installing service worker. We can include an install event listener in the service worker to perform some task when the service worker installs. For instance, during the install, service workers can precache parts of a web app so that it loads instantly the next time a user opens it (see caching the application shell). So, after that first load, you're going to benefit from instant repeat loads and your time to interactivity is going to be even better in those cases. An example of an installation event listener looks like this:

service-worker.js

```
// Listen for install event, set callback
self.addEventListener('install', function(event) {
   // Perform some task
});
```

Activation

Once a service worker has successfully installed, it transitions into the activation stage. If there are any open pages controlled by the previous service worker, the new service worker enters a waiting state. The new service worker only activates when there are no longer any pages loaded that are still using the old service worker. This ensures that only one version of the service worker is running at any given time.

When the new service worker activates, an activate event is triggered in the activating service worker. This event listener is a good place to clean up outdated caches (see the Offline Cookbook for an example).

service-worker.js

```
self.addEventListener('activate', function(event) {
  // Perform some task
});
```

Once activated, the service worker controls all pages that load within its scope, and starts listening for events from those pages. However, pages in your app that were loaded before the service worker activation will not be under service worker control. The new service worker will only take over when you close and reopen your app, or if the service worker calls **clients.claim()**. Until then, requests from this page will not be intercepted by the new service worker. This is intentional as a way to ensure consistency in your site.

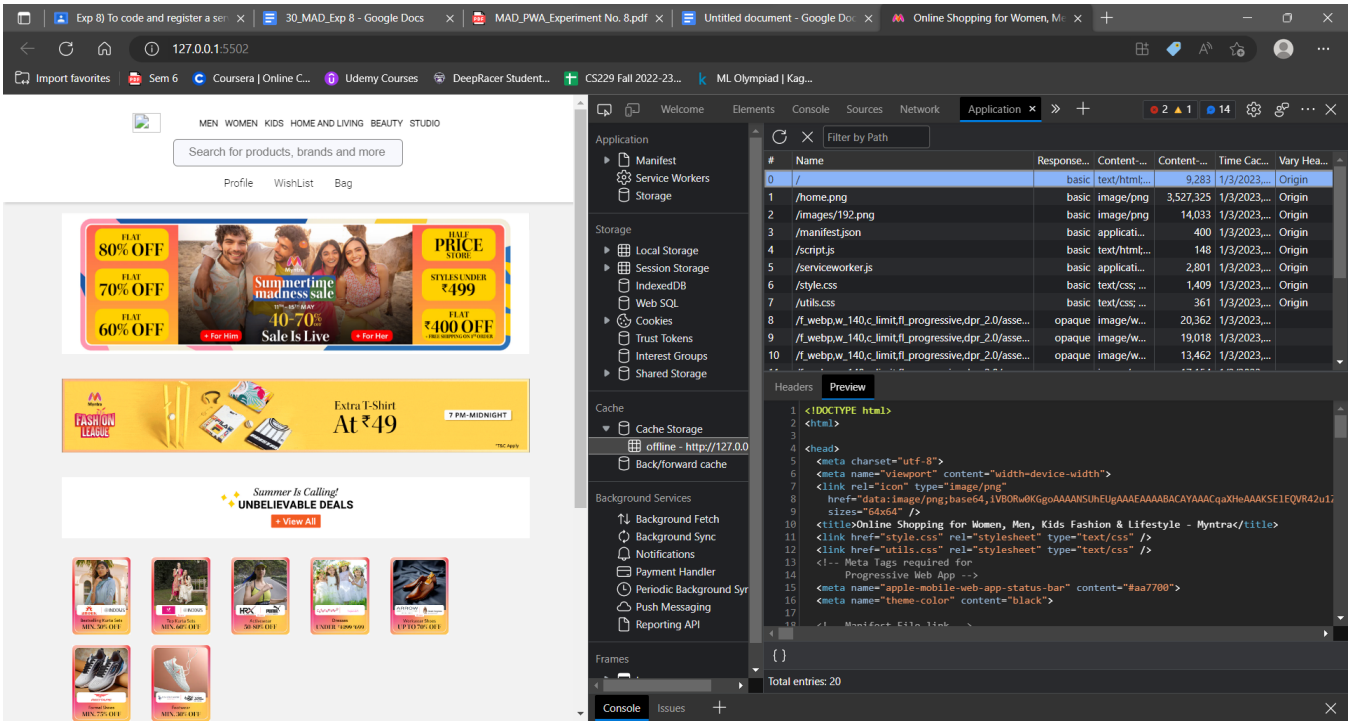
Code:

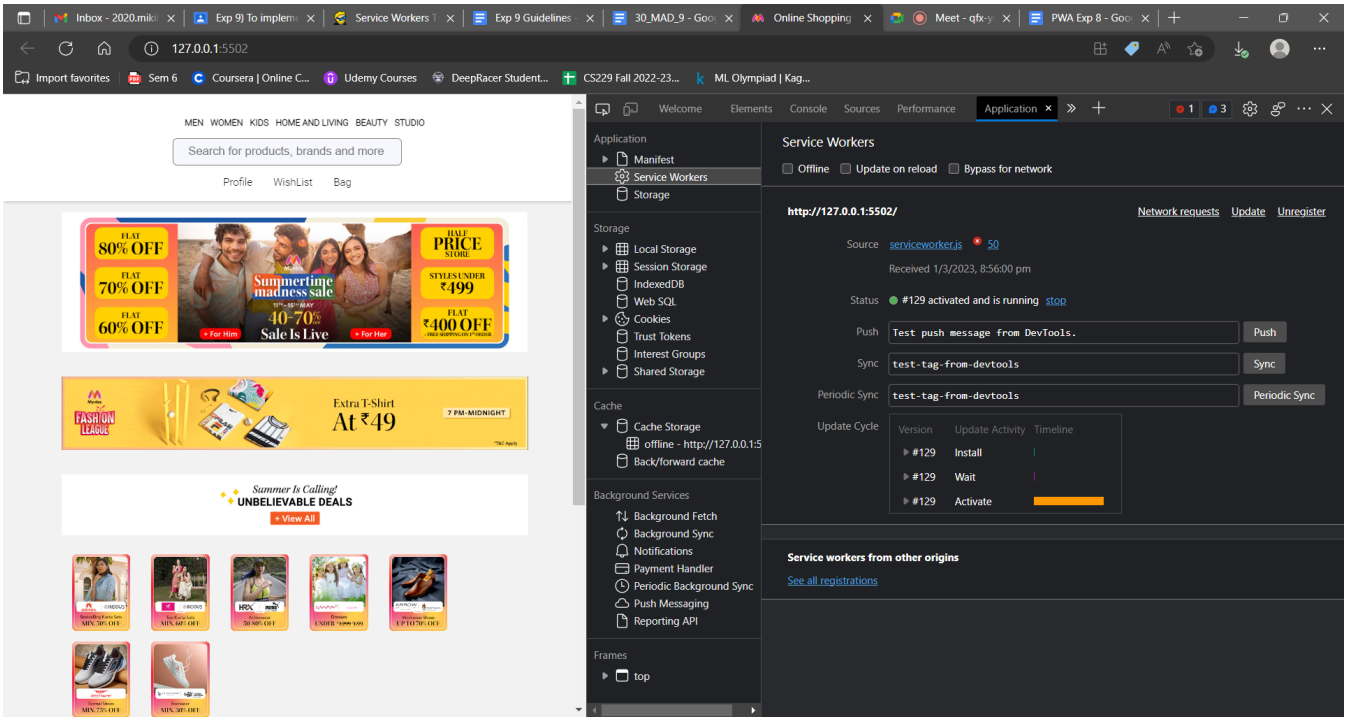
Service-worker.js

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

var preLoad = function () {
   return caches.open("offline").then(function (cache) { // caching index and important routes
      return cache.addAll(['/','/images/*']);
   });
};

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("index.html");
            } else {
                return matching;
            }
        });
    });
}
```

Output:





Conclusion-

Thus we have written code and registered a service worker, and completed the installation and activation process for a new service worker for the E-commerce PWA.