**120A3051**

**Shreya Idate**

**Batch: E3**

# EXPERIMENT 9

**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:

### Progressive web apps (PWAs)

Progressive Web Apps (PWAs) are web apps that use service workers, manifests, and other web-platform features in combination with progressive enhancement to give users an experience on par with native apps.

PWAs are web apps developed using a number of specific technologies and standard patterns to allow them to take advantage of both web and native app features. For example, web apps are more discoverable than native apps; it's a lot easier and faster to visit a website than to install an application, and you can also share web apps by sending a link.

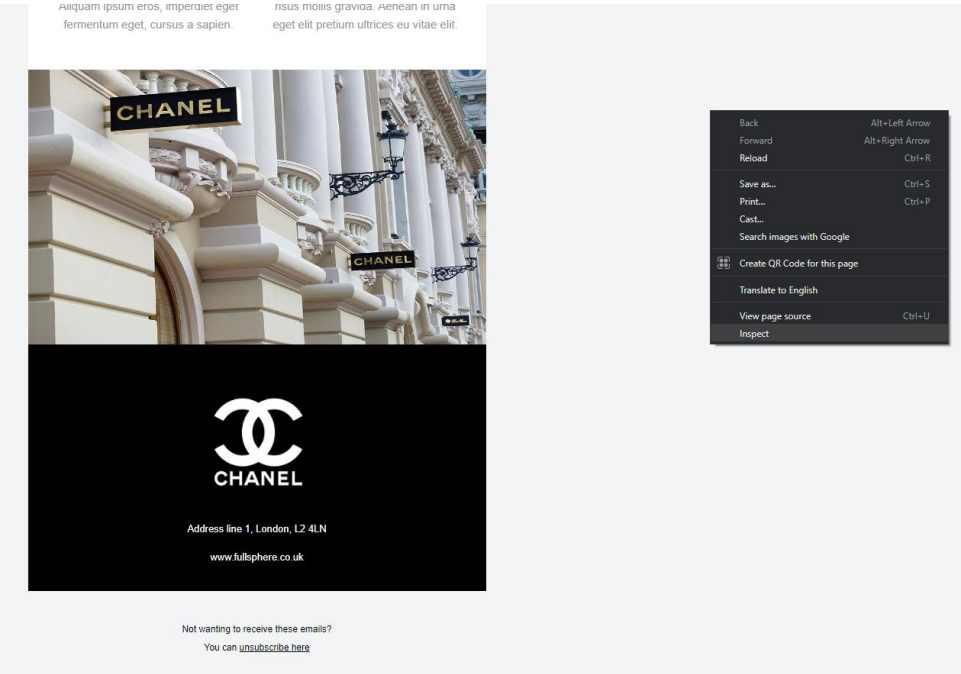
On the other hand, native apps are better integrated with the operating system and therefore offer a more seamless experience for the users. You can install a native app so that it works offline, and users love tapping their icons to easily access their favorite apps, rather than navigating to it using a browser.

PWAs give us the ability to create web apps that can enjoy these same advantages.

**What makes an app a PWA?**

There are some key principles a web app should try to observe to be identified as a PWA. It should be:

- Discoverable, so the contents can be found through search engines.
- Installable, so it can be available on the device's home screen or app launcher.
- Linkable, so you can share it by sending a URL.
- Network independent, so it works offline or with a poor network connection.
- Progressively enhanced, so it's still usable on a basic level on older browsers, but fully-functional on the latest ones.
- Re-engageable, so it's able to send notifications whenever there's new content available.
- Responsively designed, so it's usable on any device with a screen and a browser—mobile phones, tablets, laptops, TVs, refrigerators, etc.
- Secure, so the connections between the user, the app, and your server are secured against any third
- parties trying to get access to sensitive data.




## Add to Home screen (A2HS):

Add to Home screen (or A2HS for short) is a feature available in modern browsers that allows a user to "install" a web app, ie. add a shortcut to their Home screen representing their favorite web app (or site) so they can subsequently access it with a single tap.

## Manifest:

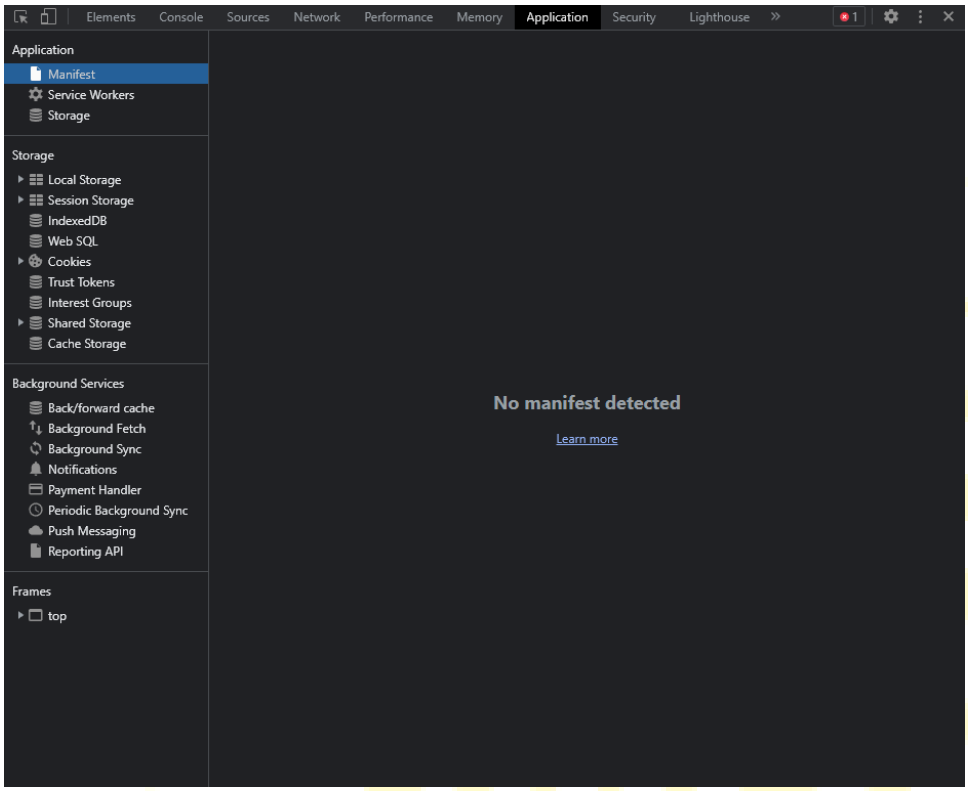
The web manifest is written in standard JSON format and should be placed somewhere inside your app directory (in the root is probably best) with the name somefilename.webmanifest. It contains multiple fields that define certain information about the web app and how it should behave.

The fields needed for A2HS are as follows:

1. **background_color:** Specifies a background color to be used in some app contexts. The most relevant one to A2HS is the splash screen displayed when the app icon on the Home screen is tapped and it first starts to load (this currently appears only when apps have been added to the Home screen by Chrome).
2. **display:** Specifies how the app should be displayed. To make it feel like a distinct app (and not just a web page), you should choose a value such as fullscreen (no UI is shown at all) or standalone (very similar, but system-level UI elements such as the status bar might be visible).
3. **icons:** Specifies icons for the browser to use when representing the app in different places (such as on the task switcher, or more important, the Home screen). We've included only one in our demo.
4. **name/short_name:** These fields provide an app name to be displayed when representing the app in different places. name provides the full app name, and short_name provides a shortened name to be used when there is insufficient space to display the full name. You are advised to provide both if your app's name is particularly long.
5. **start_url:** Provides a path to the asset that should be loaded when the added-to-Home screen app is launched. Note that this has to be a relative URL pointing to the site index, relative to the url of the manifest. Also, be aware that Chrome requires this before it will display the install banner, whereas Firefox doesn't require it for showing the home-with-a-plus icon.

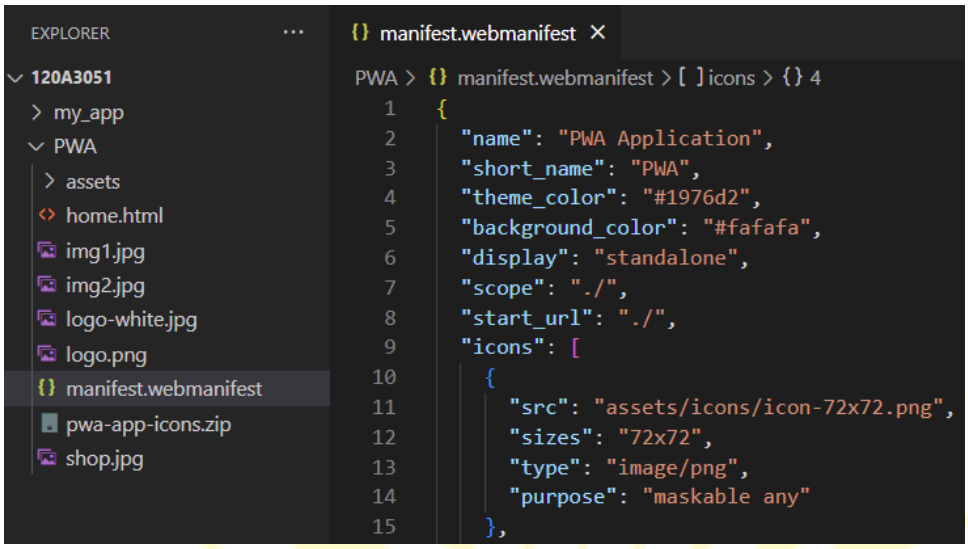
**Steps to create a PWA and A2HS:**

1. Open your webpage / website on a browser and press Ctrl+Shift+i to explore Dev Tools or Right click on your webpage and select 'Inspect'. Go to 'Application' tab that you may need to add for the first time.




2. Create an assets folder and add icons. To do so, go to https://tools.crawlink.com/tools/pwa-icon-generator/ and upload any png image and generate its PWA Icons Bundle. You can include more sizes if you want; Android will choose the most appropriate size for each different use case. You could also decide to include different types of icons so devices can use the best one they are able to. Copy the json code generated from manifest and modify as shown in 4

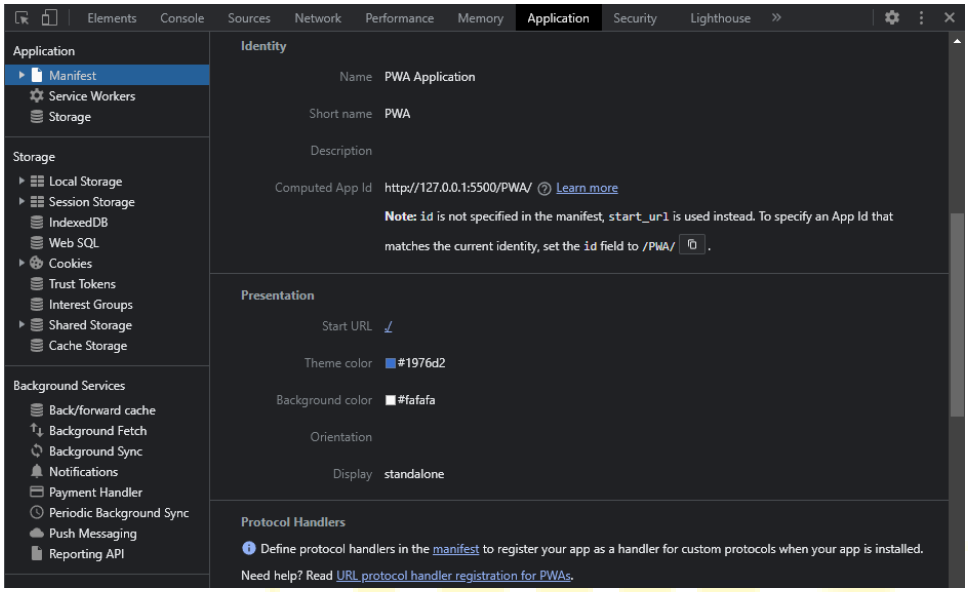
3. Create a manifest.webmanifest file in your project directory and add the foll. Json code:

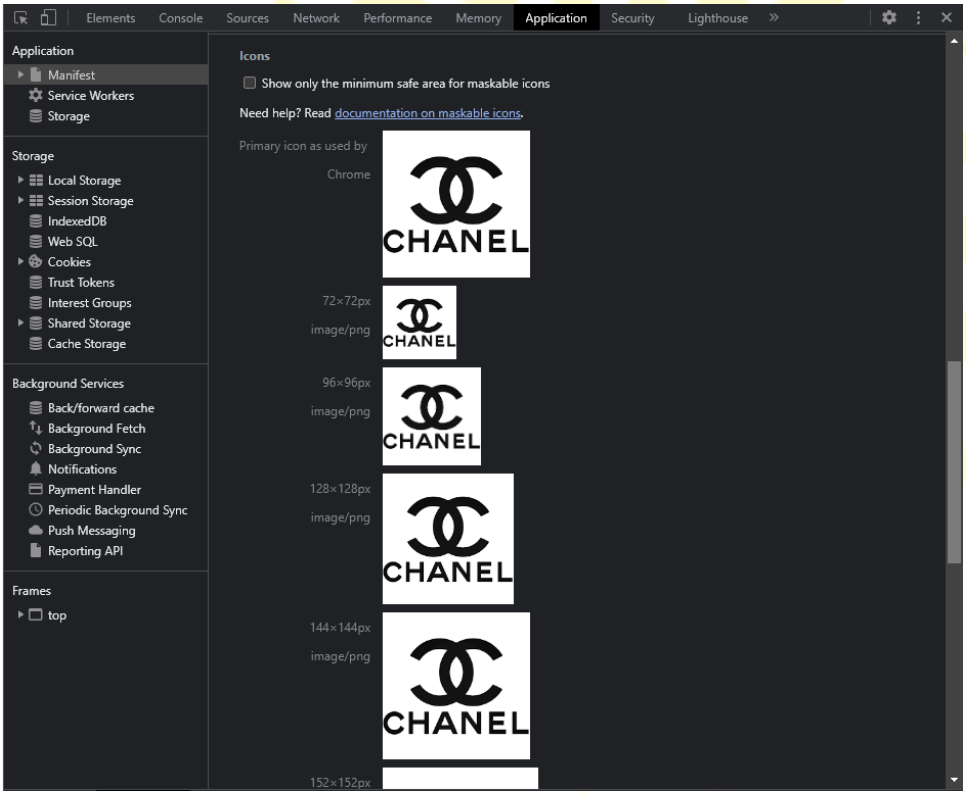



```
{
  "name": "PWA Application",
  "short_name": "PWA",
  "theme_color": "#1976d2",
  "background_color": "#fafafa",
  "display": "standalone",
  "scope": "./",
  "start_url": "./",
  "icons": [
    {
      "src": "assets/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "assets/icons/icon-96x96.png",
      "sizes": "96x96",
      "type": "image/png",
      "purpose": "any"
    },
```

```
    {
      "src": "assets/icons/icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "assets/icons/icon-144x144.png",
      "sizes": "144x144",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "assets/icons/icon-152x152.png",
      "sizes": "152x152",
      "type": "image/png",
      "purpose": "any"
    },
```

```
    {
      "src": "assets/icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "assets/icons/icon-384x384.png",
      "sizes": "384x384",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "assets/icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any"
    }
  ]
}
```




4. Link the HTML to the manifest : To finish setting up your manifest, you need to reference it from the HTML of your application's home page:

<link rel="manifest" href="manifest.webmanifest">

```
<head>
    <link rel="manifest" href="manifest.webmanifest">
    <link rel="service" href="sw.js">
```

**Adding the service worker:**

Service workers essentially act as proxy servers that sit between web applications, the browser, and the network (when available). They are intended, among other things, to enable the creation of effective offline experiences, intercept network requests and take appropriate action based on whether the network is available, and update assets residing on the server. They will also allow access to push notifications and background sync APIs.

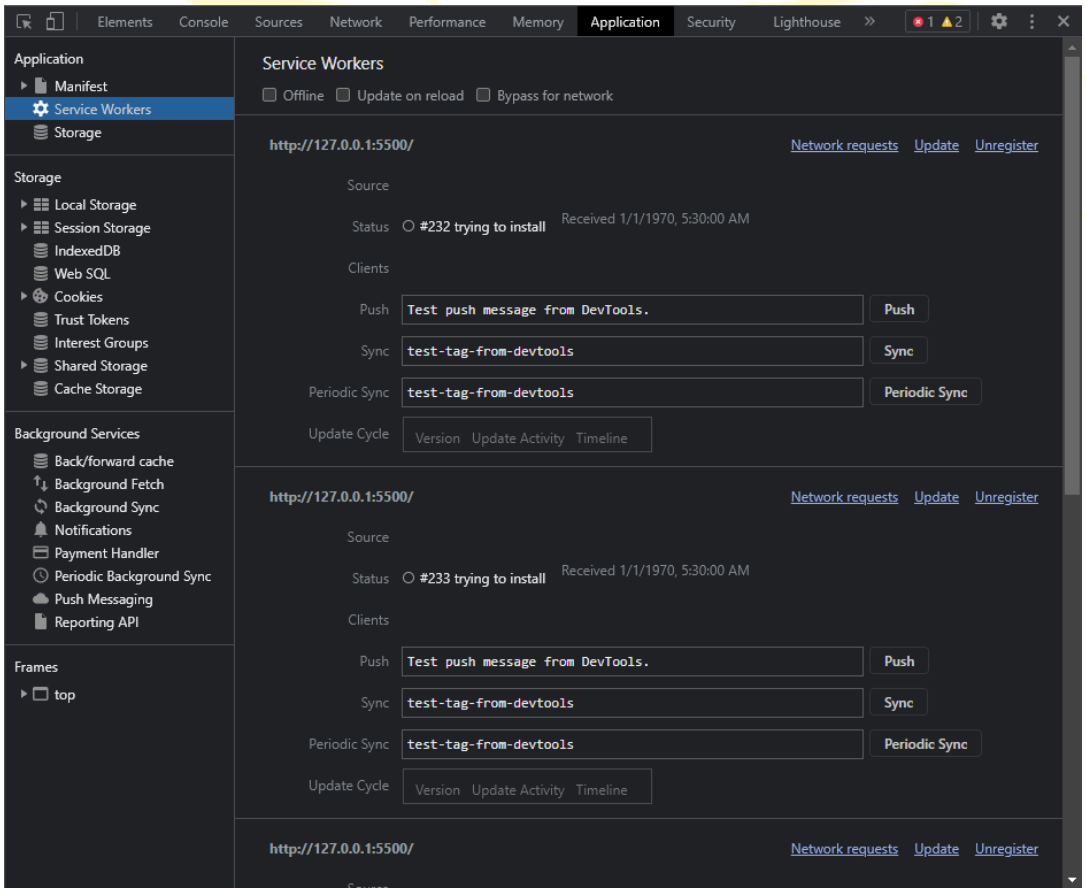
A service worker is an event-driven worker registered against an origin and a path. It takes the form of a JavaScript file that can control the web-page/site that it is associated with, intercepting and modifying navigation and resource requests, and caching resources in a very granular fashion to give you complete control over how your app behaves in certain situations (the most obvious one being when the network is not available).

To register / create a service worker, go to https://developers.google.com/web/fundamentals/primers/service-workers, and copy the code for service worker stepwise as shown next.

```
<script>
    // Don't register the service worker
    // until the page has fully loaded
    window.addEventListener('load', () => {
        // Is service worker available?
        if ('serviceWorker' in navigator) {
            navigator.serviceWorker.register('/sw.js').then(() => {
                console.log('Service worker registered!');
            }).catch((error) => {
                console.warn('Error registering service worker:');
                console.warn(error);
            });
        }
    });
</script>
```

```
<script>
    if ("serviceWorker" in navigator) {
        navigator.serviceWorker
            .register("/sw.js")
            .then((registration) => {
                registration.addEventListener("updatefound", () => {
                    // If updatefound is fired, it means that there's
                    // a new service worker being installed.
                    const installingWorker = registration.installing;
                    console.log(
                        "A new service worker is being installed:",
                        installingWorker
                    );

                    // You can listen for changes to the installing service worker's
                    // state via installingWorker.onstatechange
                });
            })
            .catch((error) => {
                console.error(`Service worker registration failed: ${error}`);
            });
    } else {
        console.error("Service workers are not supported.");
    }
</script>
```



To install and activate Servive worker to enable A2HS feature:

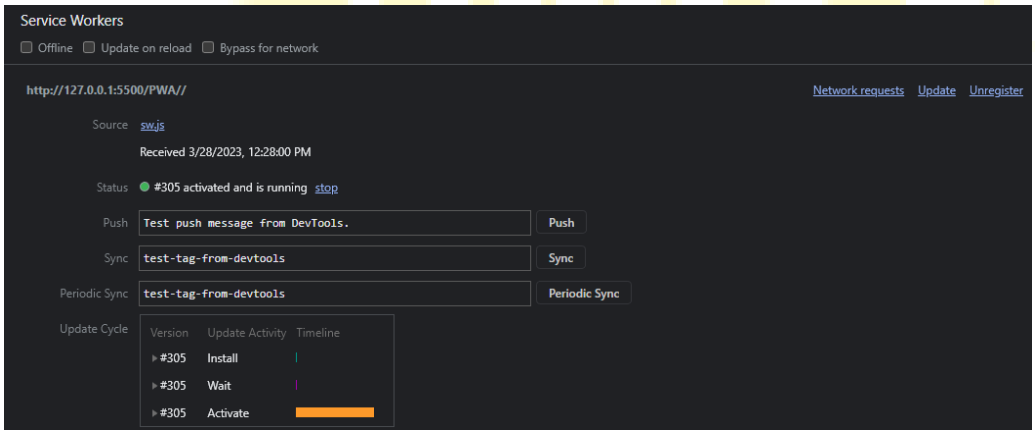
home.html 2 | sw.js | manifest.webmanifest

PWA > sw.js > urlsToCache

```
var CACHE_NAME = 'my-site-cache-v1';
var urlsToCache = [
    '/',
    'home.html',
    /* '/styles/main.css',
    '/script/main.js' */
];

self.addEventListener('install', function (event) {
    // Perform install steps
    event.waitUntil(
        caches.open(CACHE_NAME)
            .then(function (cache) {
                console.log('Opened cache');
                return cache.addAll(urlsToCache);
            })
    )
});


self.addEventListener('fetch', function (event) {
    event.respondWith(
        caches.match(event.request)
            .then(function (response) {
                // Cache hit - return response
                if (response) {
                    return response;
                }
                return fetch(event.request);
            }
            )
    );
});
```




**CONCLUSION:** Hence we have successfully registered a service worker and completed the installation and activation process for a new service worker for the E-commerce PWA.