

Experiment No:7

Aim:

To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.

Theory:

Regular Web Apps are traditional websites accessed through a browser, relying on a server for most functionality and data processing. They require an active internet connection and do not offer offline access. These apps do not provide features like push notifications or access to device hardware. Additionally, they lack a manifest file defining metadata or the ability to be installed on a device's home screen.

On the other hand, Progressive Web Apps (PWAs) offer a more app-like experience. They are built with modern web technologies like Service Workers, Web App Manifests, and HTTPS. PWAs are designed with responsive layouts, adapting to various screen sizes and orientations. They can work offline or with a poor internet connection, using Service Workers to cache content. PWAs can also send push notifications, enabling engagement even when the app is not open. They have a manifest file (manifest.json) that defines metadata such as the app's name, icons, colors, and start URL. Users can "install" a PWA on their device's homescreen, making it accessible like a native app, with an icon and full-screen mode. PWAs are discoverable and can be indexed by search engines, improving visibility. They provide a seamless, fast, and reliable user experience, leading to higher user engagement and retention.

In terms of differences, Regular Web Apps typically rely on an active internet connection to function properly and have limited functionality when offline. In contrast, PWAs use Service Workers to cache content, allowing them to work offline or in low-connectivity situations. Updates made while offline are synchronized when the device reconnects to the internet. Installation differs as well, with Regular Web Apps accessed by typing the URL into a web browser and lacking an option for installation on the device's homescreen. PWAs, however, can be "installed" on the user's device, creating an icon on the homescreen, and launching in full-screen mode, resembling native mobile apps. Additionally, Regular Web Apps lack the ability to send push notifications, while PWAs have the capability to send push notifications, even when the app is not actively open.

In terms of metadata and manifest, Regular Web Apps do not have a manifest file (manifest.json) defining metadata, while PWAs use a manifest.json file to define metadata about the app. Developers can specify the app's name, icons for different device sizes, background color, theme color, and more. This metadata allows the PWA to appear more like a native app when installed on the homescreen. Performance and optimization also differ, with Regular Web Apps' performance based on the browser and device capabilities, while PWAs are optimized for

performance and reliability. They offer a faster, more responsive, and smoother user experience, especially on mobile devices, and can be added to the user's home screen, launching quickly and providing a native-like feel. Finally, in terms of discoverability and indexing, Regular Web Apps may have limited discoverability by search engines and may not appear in app stores or be easily found by users searching for similar apps, while PWAs are fully discoverable by search engines, improving visibility, and can be indexed and ranked in search results, similar to regular websites. Users searching for related topics may discover PWAs through search engine results.

Implementation:

The below steps have to be followed to create a progressive web application:

Step 1: Create an HTML page that would be the starting point of the application. This HTML will contain a link to the file named manifest.json. This is an important file that will be created in the next step.

Step 2: Create a manifest.json file in the same directory. This file contains information about the web application. Some basic information includes the application name, starting URL, theme color, and icons. All the information required is specified in the JSON format. The source and size of the icons are also defined in this file.

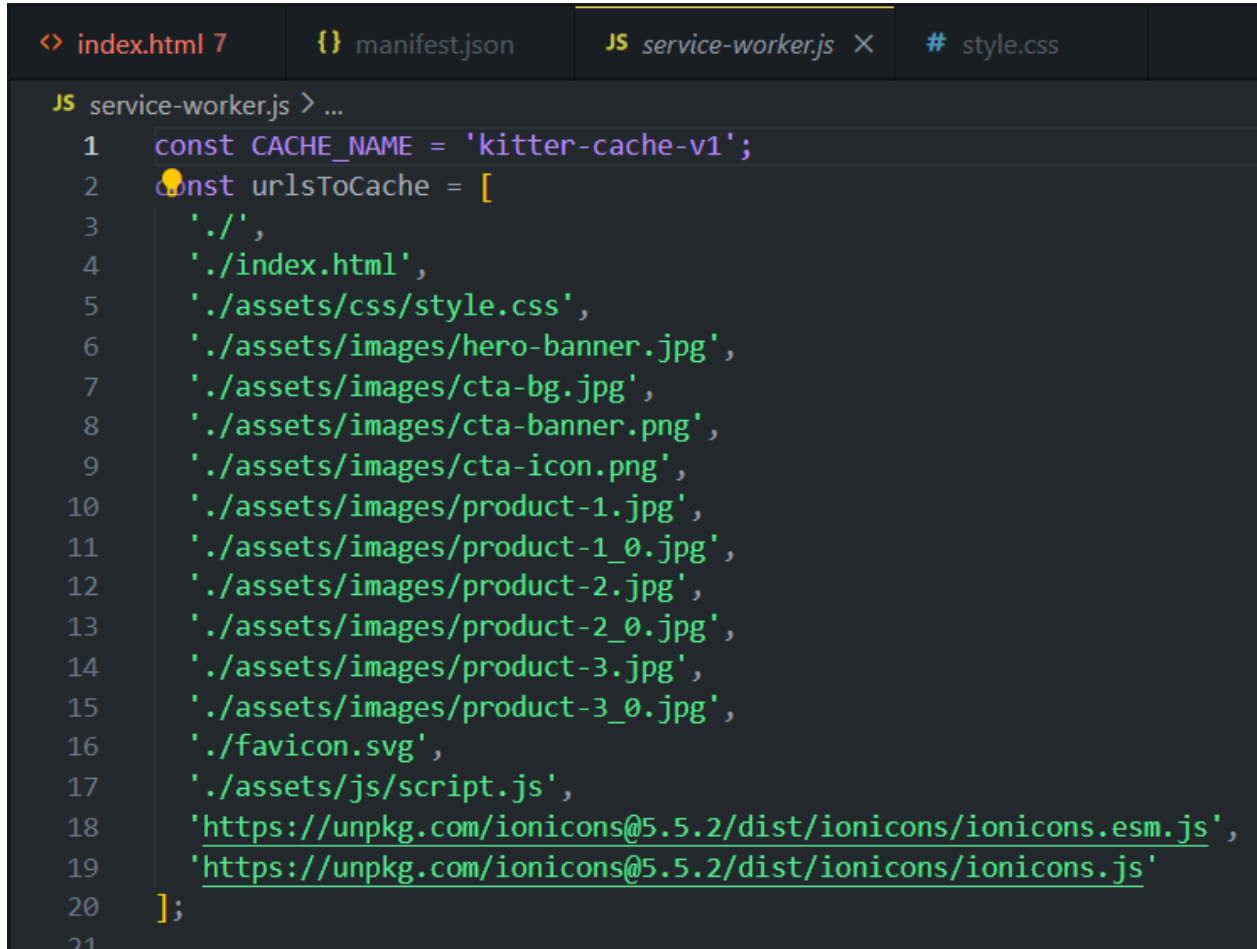
```
{
  "name": "Fashionify - High-Quality Clothing",
  "short_name": "Fashionify",
  "description": "High Quality Clothing eCommerce Website",
  "icons": [
    {
      "src": "./favicon.svg",
      "type": "image/svg+xml",
      "sizes": "any"
    }
  ],
  "start_url": "/index.html",
  "display": "standalone",
  "theme_color": "#ffffff",
  "background_color": "#ffffff"
}
```

Step 3: Create a new folder named images and place all the icons related to the application in that folder. It is recommended to have the dimensions of the icons at least 192 by 192 pixels and 512

by 512 pixels. The image name and dimensions should match that of the manifest file.

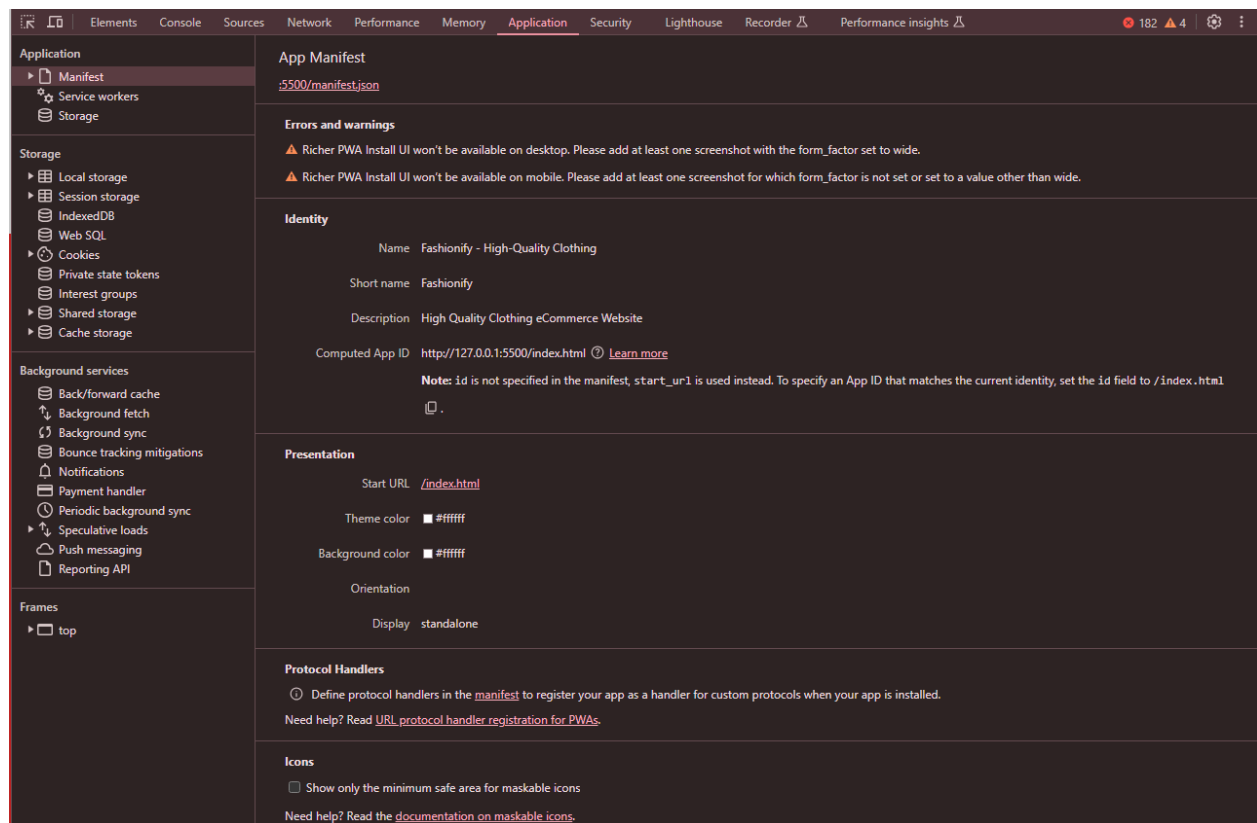
Step 4: Serve the directory using a live server so that all files are accessible.

Step 5: Open the index.html file in Chrome navigate to the Application Section in the Chrome Developer Tools. Open the manifest column from the list.



```
<> index.html 7    {} manifest.json    JS service-worker.js X    # style.css

JS service-worker.js > ...
1  const CACHE_NAME = 'kitter-cache-v1';
2  const urlsToCache = [
3    './',
4    './index.html',
5    './assets/css/style.css',
6    './assets/images/hero-banner.jpg',
7    './assets/images/cta-bg.jpg',
8    './assets/images/cta-banner.png',
9    './assets/images/cta-icon.png',
10   './assets/images/product-1.jpg',
11   './assets/images/product-1_0.jpg',
12   './assets/images/product-2.jpg',
13   './assets/images/product-2_0.jpg',
14   './assets/images/product-3.jpg',
15   './assets/images/product-3_0.jpg',
16   './favicon.svg',
17   './assets/js/script.js',
18   'https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.esm.js',
19   'https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.js'
20 ];
21
```

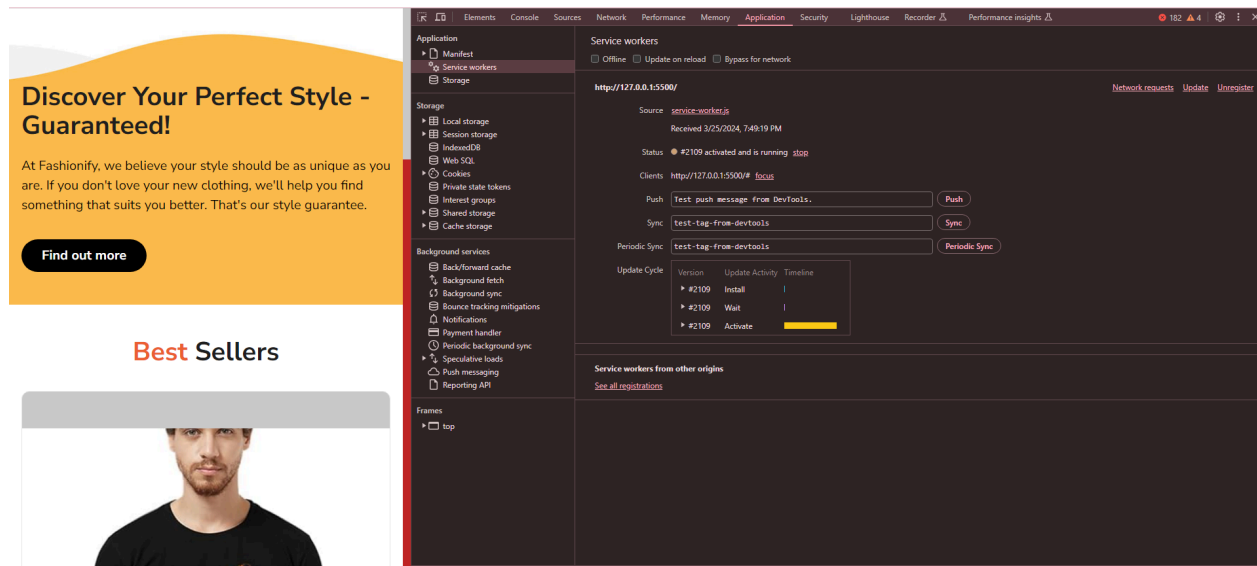


Step 6: Under the installability tab, it would show that no service worker is detected. We will need to create another file for the PWA, that is, serviceworker.js in the same directory. This file handles the configuration of a service worker that will manage the working of the application.

Step 7: The last step is to link the service worker file to index.html. This is done by adding a short JavaScript script to the index.html created in the above steps. Add the below code inside the script tag in index.html.

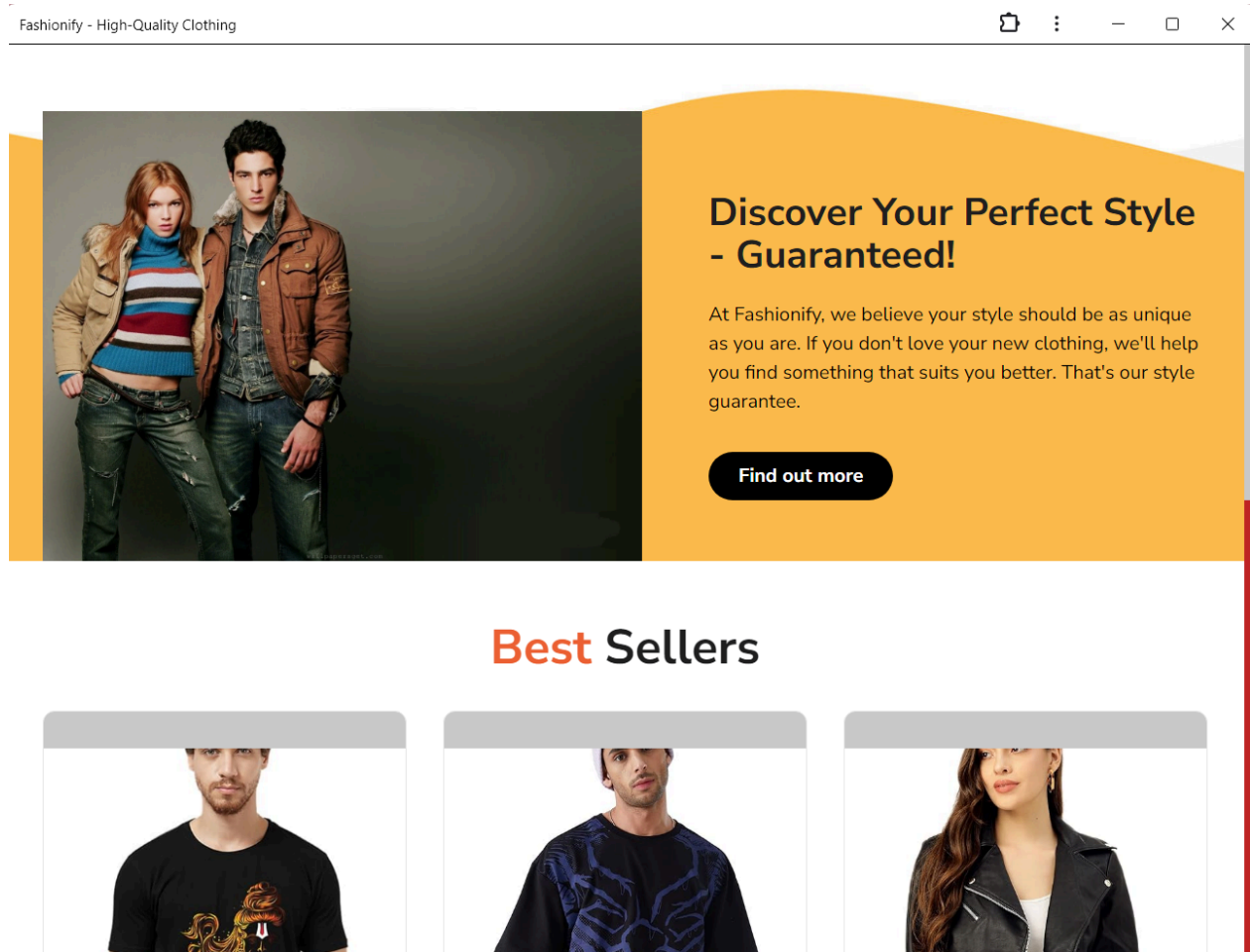
```
<script>
  if ('serviceWorker' in navigator) {
    window.addEventListener('load', () => {
      navigator.serviceWorker.register('/service-worker.js')
        .then(registration => {
          console.log('Service Worker registered with scope:', registration.scope);
        })
        .catch(error => {
          console.error('Service Worker registration failed:', error);
        });
    });
  }
</script>

</body>
</html>
```



Installing the application:

- Navigating to the Service Worker tab, we see that the service worker is registered successfully and now an install option will be displayed that will allow us to install our app.
- Click on the install button to install the application. The application would then be installed, and it would be visible on the desktop.
- For installing the application on a mobile device, the Add to Home screen option in the mobile browser can be used. This will install the application on the device.



Conclusion :

This experiment involved creating a straightforward HTML E-commerce home page showcasing products with images, titles, prices, descriptions, and "Add to Cart" buttons. This foundational layout offers a starting point for developing a more robust E-commerce platform, with the potential for expansion into categories, navigation.