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 App:
- A regular web app is a traditional website accessed through a browser.
- It relies on a server for most of its functionality and data processing.
- It does not necessarily offer offline access, relying on an active internet connection.
- It may not provide features like push notifications or access to device hardware.
- It does not have a manifest file defining its metadata or the ability to be installed on a device's homescreen.

Progressive Web App (PWA):
- A Progressive Web App (PWA) is a type of web application that provides a more app-like experience for users.
- It is built with modern web technologies like Service Workers, Web App Manifests, and HTTPS.
- PWAs offer responsive design, 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 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.

Differences: Offline Functionality:
- Regular Web Apps:
- Typically rely on an active internet connection to function properly.
- When the user loses connectivity, they may see error messages or a lack of new data.

- The app's functionality is limited when offline, as it cannot access previously loaded content. ● Progressive Web Apps (PWAs):
- Use Service Workers to cache content, allowing them to work offline or in low-connectivity situations.
- Users can continue to browse content, view pages, and interact with the app even without an internet connection.
- Updates made while offline are synchronized when the device reconnects to the internet.

Installation:
- Regular Web Apps:
- Accessed by typing the URL into a web browser.
- Do not have an option for installation on the device's homescreen.
- Progressive Web Apps (PWAs): ● Can be "installed" on the user's device, creating an icon on the homescreen.
- This installation is done through a browser prompt or a manual option from the browser menu.
- Once installed, PWAs launch in full-screen mode, resembling native mobile apps.

Push Notifications:
- Regular Web Apps:
- Lack the ability to send push notifications to users.
- Engagement with users outside of the app is limited.
- Progressive Web Apps (PWAs):
- Have the capability to send push notifications, even when the app is not actively open. ● This feature helps in re-engaging users, announcing updates, promotions, or relevant information.

Metadata and Manifest:
- Regular Web Apps:
- Do not have a manifest file (manifest.json) defining metadata.
- Metadata like icons, colors, and start URLs are typically not specified.
- Progressive Web Apps (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:
- Regular Web Apps:

- Performance is based on the browser and device capabilities.
- May not always provide a smooth, app-like experience due to varying browser support.
- Progressive Web Apps (PWAs):
- Optimized for performance and reliability.
- Utilize modern web technologies such as Service Workers and efficient caching strategies.
- Offer a faster, more responsive, and smoother user experience, especially on mobile devices. ● Can be added to the user's homescreen, launching quickly and providing a native-like feel.
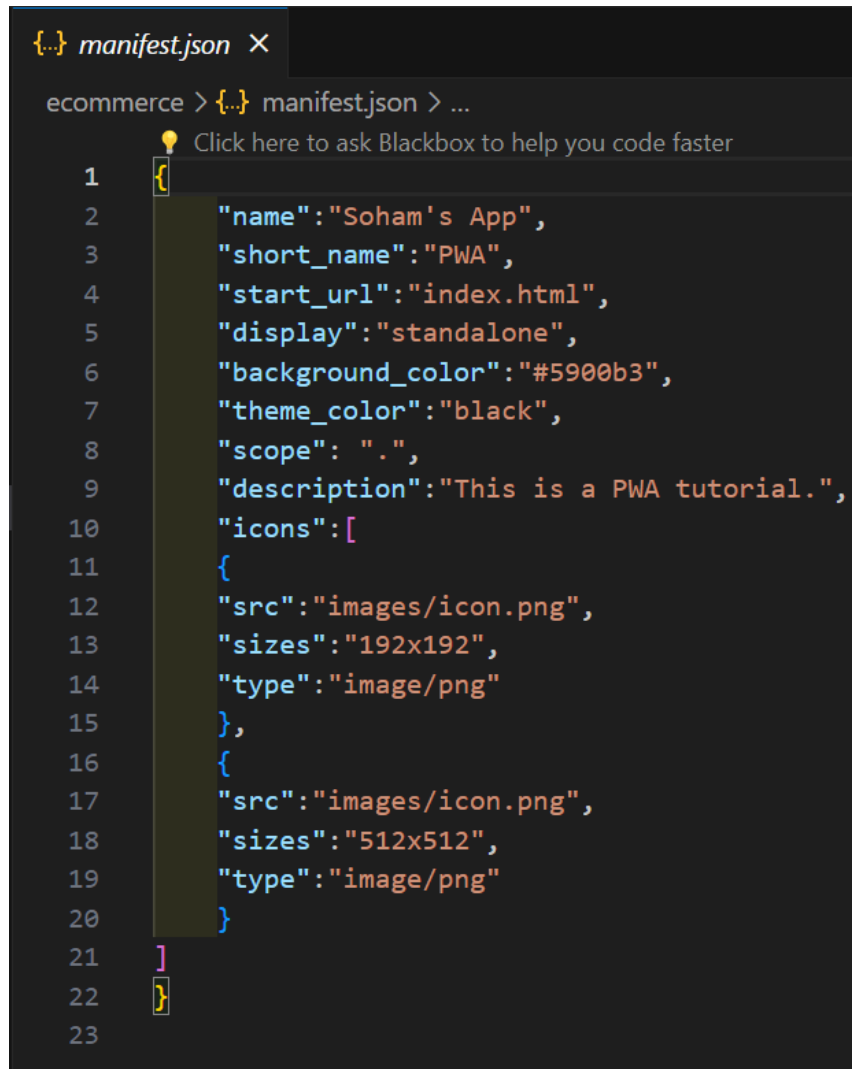
Discoverability and Indexing:
- Regular Web Apps:
- Discoverability by search engines can be limited.
- May not appear in app stores or be easily found by users searching for similar apps.
- Progressive Web Apps (PWAs):
- Fully discoverable by search engines, improving visibility.
- Can be indexed and 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.

```
{..} manifest.json ✕

ecommerce > {..} manifest.json > ...
          💡 Click here to ask Blackbox to help you code faster
   1   {
   2       "name":"Soham's App",
   3       "short_name":"PWA",
   4       "start_url":"index.html",
   5       "display":"standalone",
   6       "background_color":"#5900b3",
   7       "theme_color":"black",
   8       "scope": ".",
   9       "description":"This is a PWA tutorial.",
  10       "icons":[
  11       {
  12       "src":"images/icon.png",
  13       "sizes":"192x192",
  14       "type":"image/png"
  15       },
  16       {
  17       "src":"images/icon.png",
  18       "sizes":"512x512",
  19       "type":"image/png"
  20       }
  21       ]
  22   }
  23
```

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.

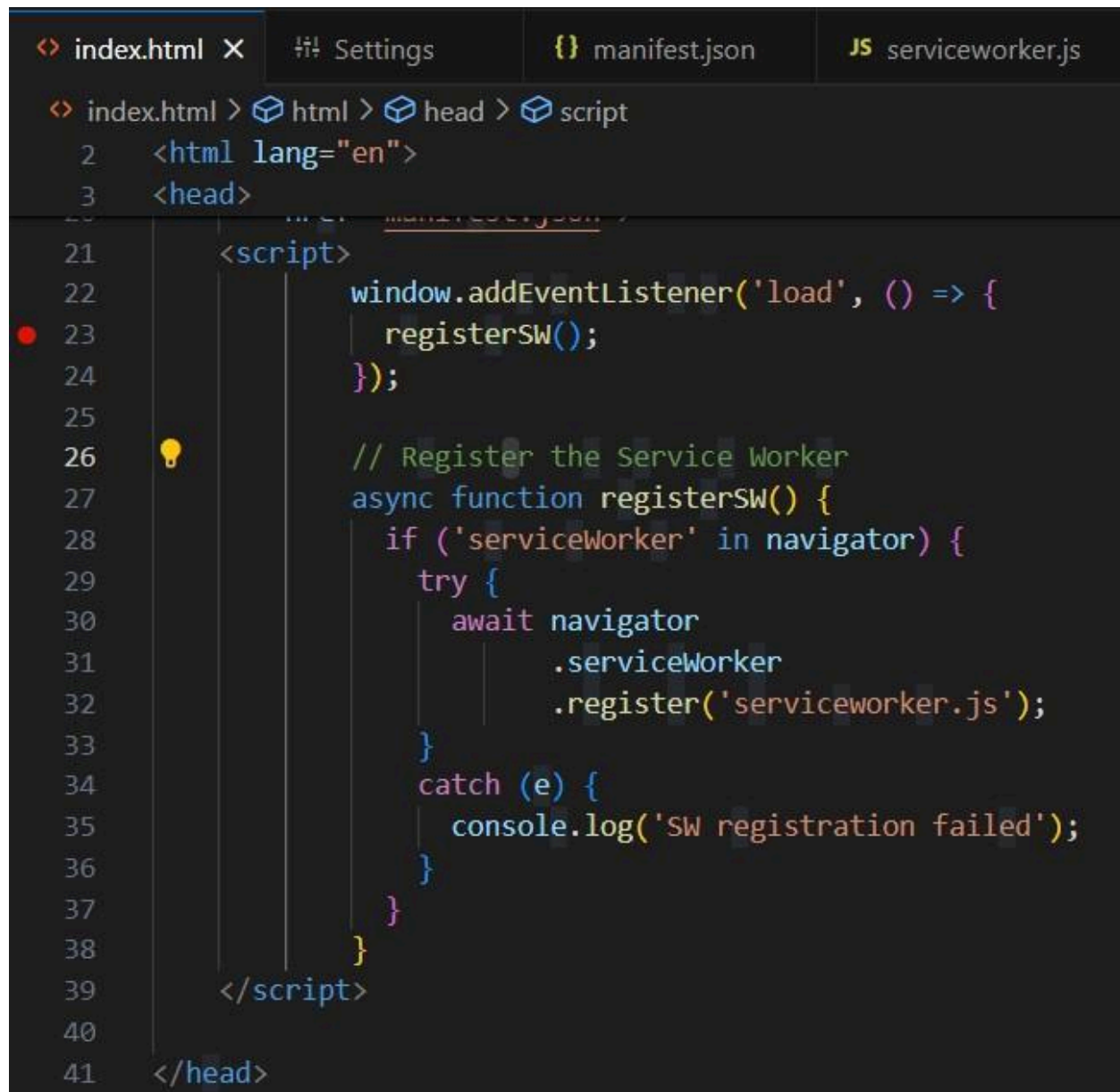Step 4: Serve the directory using a live server so that all files are accessible.
Step 5: Open the index.html file in Chrome navigate to the Application Section in the Chrome Developer Tools. Open the manifest column from the list.
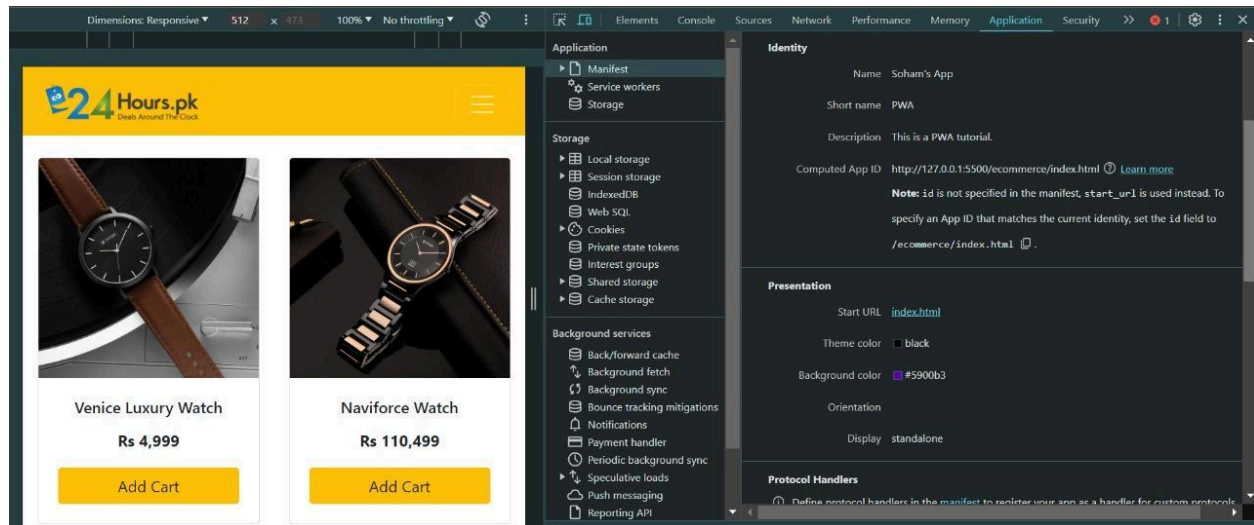
Step 6: Under the installability tab, it would show that no service worker is detected. We will need to create another file for the PWA, that is, serviceworker.js in the same directory. This file handles the configuration of a service worker that will manage the working of the application.

```js
var staticCacheName = "pwa";

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);
    })
  );
});
```

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.

```
<> index.html  X        Settings        {} manifest.json        JS serviceworker.js

<> index.html > html > head > script
  2      <html lang="en">
  3      <head>
 21          <script>
 22                  window.addEventListener('load', () => {
 23                    registerSW();
 24                  });
 25
 26                  // Register the Service Worker
 27                  async function registerSW() {
 28                    if ('serviceWorker' in navigator) {
 29                      try {
 30                        await navigator
 31                              .serviceWorker
 32                              .register('serviceworker.js');
 33                      }
 34                      catch (e) {
 35                        console.log('SW registration failed');
 36                      }
 37                    }
 38                  }
 39          </script>
 40
 41      </head>
```
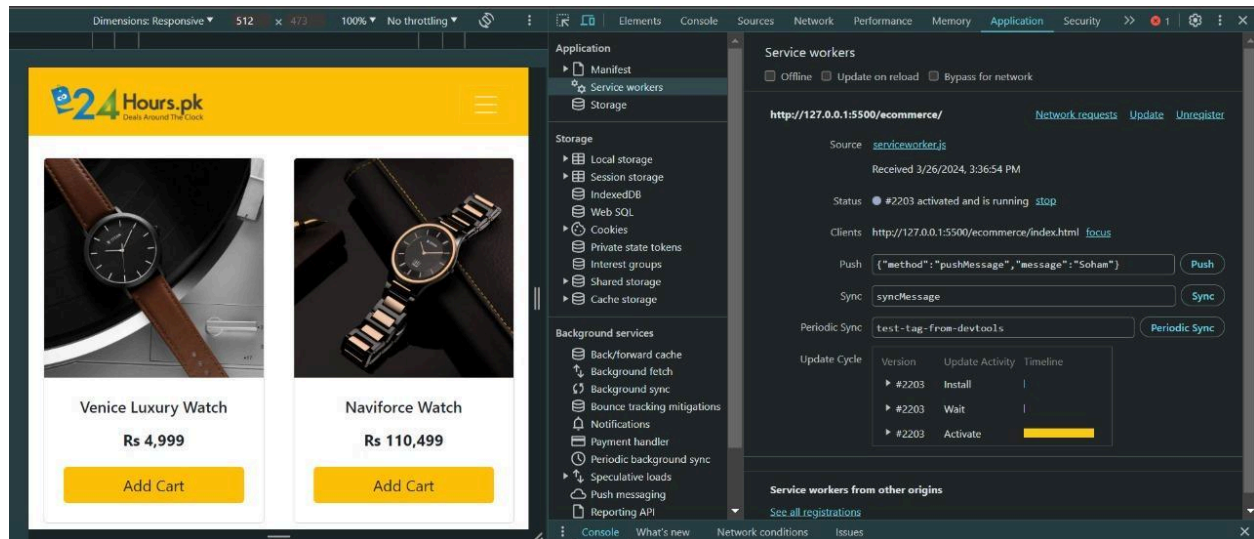
Installing the application:

● 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.
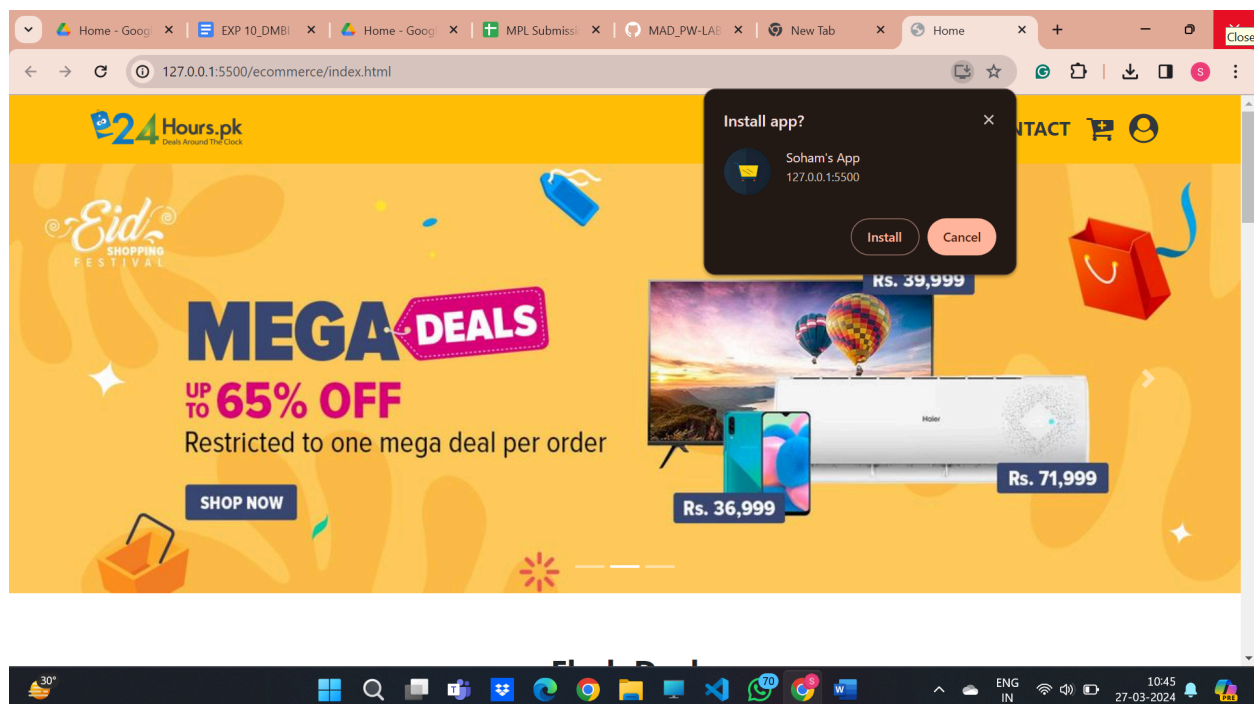
Conclusion :

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.