Aniket Mahajan
D15B 33

Experiment No : 07

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

Theory:

Progressive Web Apps (PWAs):

It refers to a type of web application that uses modern web capabilities to provide a user experience similar to that of native mobile apps. PWAs are built using web technologies such as HTML, CSS, and JavaScript but are designed to work on any platform that uses a standards-compliant browser, including desktop and mobile devices.

Regular web app

It refers to an application that is accessed and used through a web browser on various devices, such as desktops, laptops, tablets, and mobile phones. Here are some key characteristics of regular web apps:

How Progressive Web Apps (PWAs) differ from regular web apps?

Progressive Web Apps (PWAs) differ from regular web apps in several key aspects, which contribute to providing a more app-like experience for users. Here's how PWAs differ from regular web apps:

Offline Functionality: PWAs can work offline or with a poor internet connection, thanks to service workers. Service workers are scripts that run in the background and can intercept network requests, enabling features like caching of assets and data. This allows PWAs to continue functioning even when the user is offline, providing a more reliable experience compared to regular web apps, which typically require an active internet connection.

App-like User Interface: PWAs are designed to provide a more immersive and app-like user experience. They can be installed on the device's homescreen and launched in full-screen mode, without the browser's address bar or navigation buttons, giving them a native app feel. Additionally, PWAs can utilize features like push notifications, enabling businesses to engage with users even when the app is not actively in use.

Improved Performance: PWAs are optimized for performance, providing faster loading times and smoother interactions compared to traditional web apps. Techniques like lazy loading, code splitting, and caching help minimize loading times and reduce the amount of data transferred over the network. This results in a more responsive and snappy user experience, especially on slower connections or low-end devices.

Discoverability and Shareability: PWAs are discoverable through web search engines, just like regular web apps. However, they can also be shared via URL links, making it easy for users to share content with others. Additionally, PWAs can leverage features like web app manifests and service worker registration scopes to provide a more integrated experience when installed on a user's device.

Cross-platform Compatibility: PWAs are designed to work across different platforms and devices, including desktops, tablets, and mobile phones. They utilize responsive design principles to adapt to various screen sizes and orientations, ensuring a consistent user experience across devices. This cross-platform compatibility makes PWAs a versatile solution for reaching users on different devices without the need to develop separate native apps for each platform.

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 would be created in the next step.

```
<body>
    <header>
        <h1>Simple E-commerce</h1>
    </header>
    <nav>
        <a href="#">Home</a>
        <a href="#">Products</a>
        <a href="#">Contact</a>
    </nav>
    <div class="container">
        <div class="product">
            <img src="images/amazon.jpeg" alt="Product 1">
            <h3>Run</h3>
            <p>₹1999</p>
            <button class="button">Add to Cart</button>
        </div>
        <div class="product">
            <img src="images/nike.jpg" alt="Product 2">
            <h3>Nike</h3>
            <p>₹2499</p>
            <button class="button">Add to Cart</button>
        </div>
        <div class="product">
            <img src="images/adidas.jpg" alt="Product 3">
            <h3>      </h3>
```

```
            <p>₹2999</p>
            <button class="button">Add to Cart</button>
        </div>
        <!-- Add more product divs as needed -->
    </div>
    <script src="service_worker.js"></script>
</body>
</html>
```

Step 2: Create a manifest.json file in the same directory. This file basically 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.

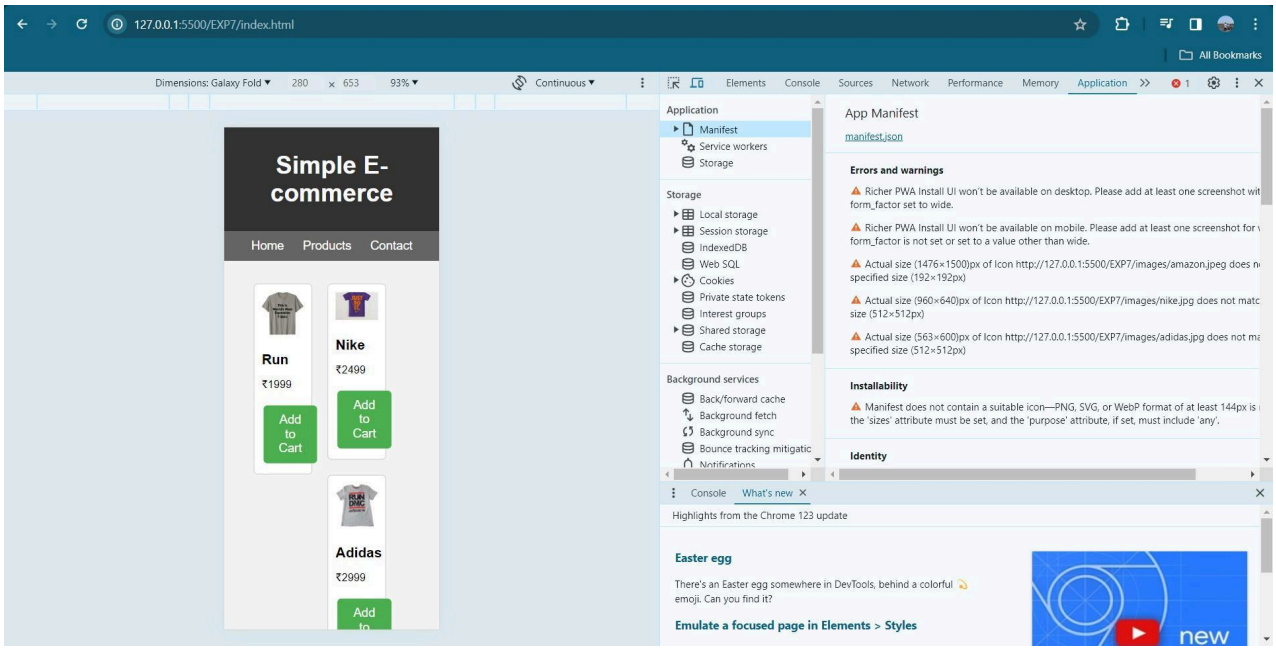
```
{
    "name": "Simple E-commerce",
    "short_name": "E-commerce",
    "description": "A simple e-commerce page showcasing products.",
    "start_url": "/",
    "display": "standalone",
    "background_color": "#f4f4f4",
    "theme_color": "#333",
    "icons": [
      {
        "src": "images/amazon.jpeg",
        "sizes": "192x192",
        "type": "image/jpeg"
      },
      {
        "src": "images/nike.jpg",
        "sizes": "512x512",
        "type": "image/jpg"
      },
      {
        "src": "images/adidas.jpg",
```

```
        "sizes": "512x512",
        "type": "image/jpg"
      }
  ]
}
```

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.

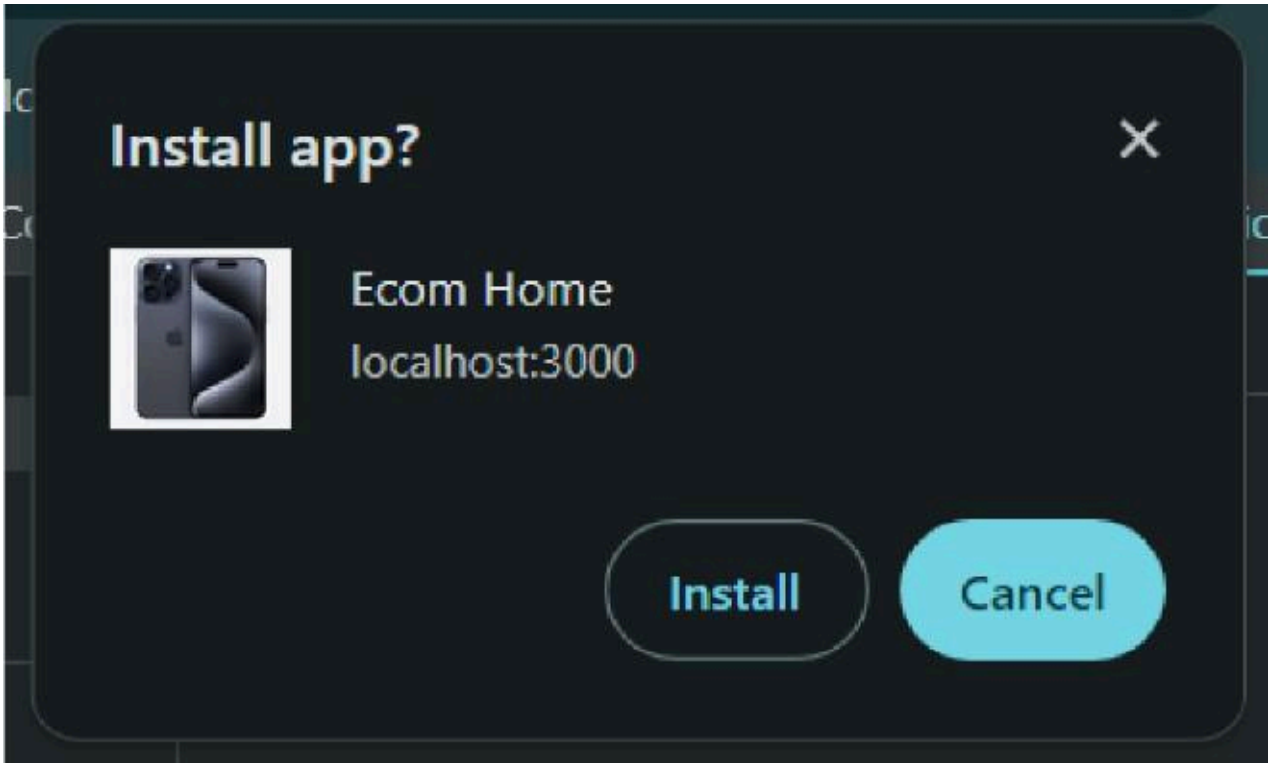
Step 7: The last step is to link the service worker file to index.html. This is done by adding a short JavaScript script to the index.html created in the above steps. Add the below code inside the script tag in index.html.

```
<script src="./app.js">
    window.addEventListener('load', () => {
        registerSW();
    });

    // Register the Service Worker
    async function registerSW() {
        if ('serviceWorker' in navigator) {
            try {
                await navigator
                    .serviceWorker
                    .register('serviceworker.js');
            }
            catch (e) {
                console.log('SW registration failed');
            }
        }
    }
</script>
```

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: Thus writing metadata for the PWA, especially for an eCommerce application, is crucial for enabling features like the "add to homescreen" functionality. By crafting a well-structured manifest.json file with accurate metadata properties such as name, description, icons, and colors, developers can enhance the accessibility and user experience of their PWAs.