

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):

Progressive Web Apps (PWAs) are web applications that leverage modern web capabilities to offer a user experience similar to native mobile apps. They are built using standard web technologies like HTML, CSS, and JavaScript but are designed to work on any platform with a standards-compliant browser, including desktop and mobile devices.

Regular web app

Regular web apps, on the other hand, are accessed and used through a web browser on various devices, such as desktops, laptops, tablets, and mobile phones. While they also use web technologies, they may lack some of the advanced features and user experience enhancements found in PWAs.

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:

1. **Offline Functionality:** PWAs can work offline or with a poor internet connection, thanks to service workers. This enables features like caching of assets and data, allowing PWAs to continue functioning even when the user is offline. Regular web apps typically require an active internet connection to function properly.
2. **App-like User Interface:** PWAs can be installed on the device's homescreen and launched in full-screen mode, giving them a native app feel. They can also utilize features like push notifications, providing a more immersive and engaging user experience compared to regular web apps.
3. **Improved Performance:** PWAs are optimized for performance, offering faster loading times and smoother interactions. Techniques like lazy loading, code splitting, and caching help minimize loading times and reduce data transfer, resulting in a more responsive user experience.
4. **Discoverability and Shareability:** PWAs are discoverable through web search engines and can be shared via URL links. This makes it easy for users to discover and share PWAs with others. Regular web apps may have limited discoverability and shareability compared to PWAs.
5. **Cross-platform Compatibility:** PWAs are designed to work across different platforms and devices, ensuring a consistent user experience. They utilize responsive design principles to adapt to various screen sizes and orientations, making them a versatile solution for reaching users on different devices.

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.

```
index.html X manifest.json script.js background.jpg style.css
index.html > html > body > main > section.featured-products > div.product-list > div.product-card > img
💡 Click here to ask Blackbox to help you code faster
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Mobile Shop - Your Ultimate Destination for Phones</title>
7     <link rel="stylesheet" href="style.css">
8     <link rel="manifest" href="manifest.json" />
9   </head>
10  <body>
11    <header>
12      <h1>Mobile Shop</h1>
13      <nav>
14        <ul>
15          <li><a href="#">Home</a></li>
16          <li><a href="#">Shop</a></li>
17          <li><a href="#">About</a></li>
18          <li><a href="#">Contact</a></li>
19        </ul>
20      </nav>
21    </header>
22
23  <main>
24    <section class="featured-products">
25      <h2>Featured Phones</h2>
26      <div class="product-list">
27        <div class="product-card">
28          
29          <h3>Iphone 15</h3>
30          <p>$499</p>
31          <button>Add to Cart</button>
```

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.

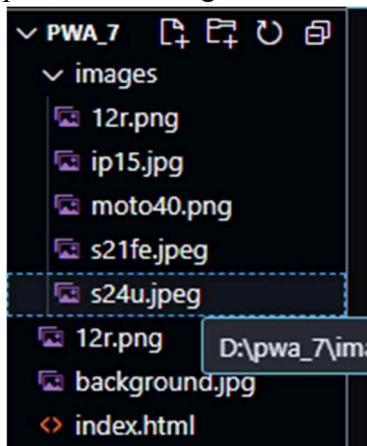
```

1  Click here to ask Blackbox to help you code faster
2  {
3      "name": "Mobile Ecommerce Homepage",
4      "short_name": "Ecommerce",
5      "start_url": "/",
6      "display": "standalone",
7      "theme_color": "#007bff",
8      "background_color": "#ffffff",
9      "icons": [
10         {
11             "src": "images/s21fe.jpeg",
12             "type": "image/png",
13             "sizes": "225x225"
14         },
15         {
16             "src": "images/s24u.jpeg",
17             "type": "image/png",
18             "sizes": "225x225"
19         },
20         {
21             "src": "images/12r.png",
22             "type": "image/png",
23             "sizes": "225x225"
24         },
25     ],
26     "description": "Homepage for our Ecommerce store.",
27     "categories": ["shopping", "business"],
28     "prefer_related_applications": false,
29     "related_applications": []
30 }

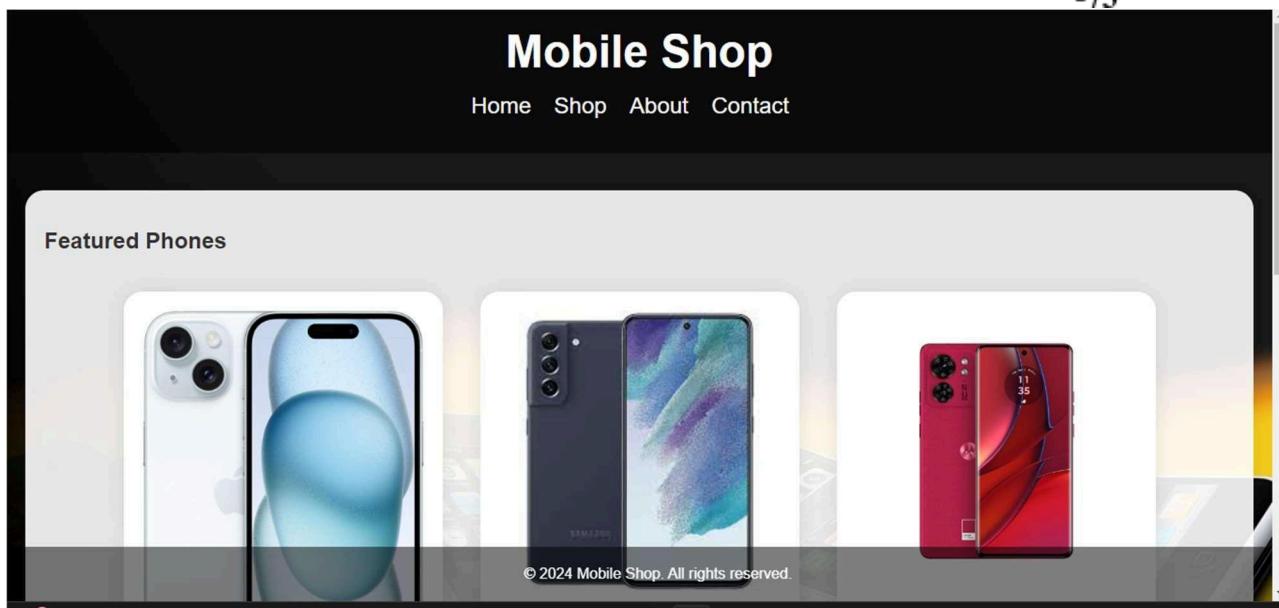
```

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.

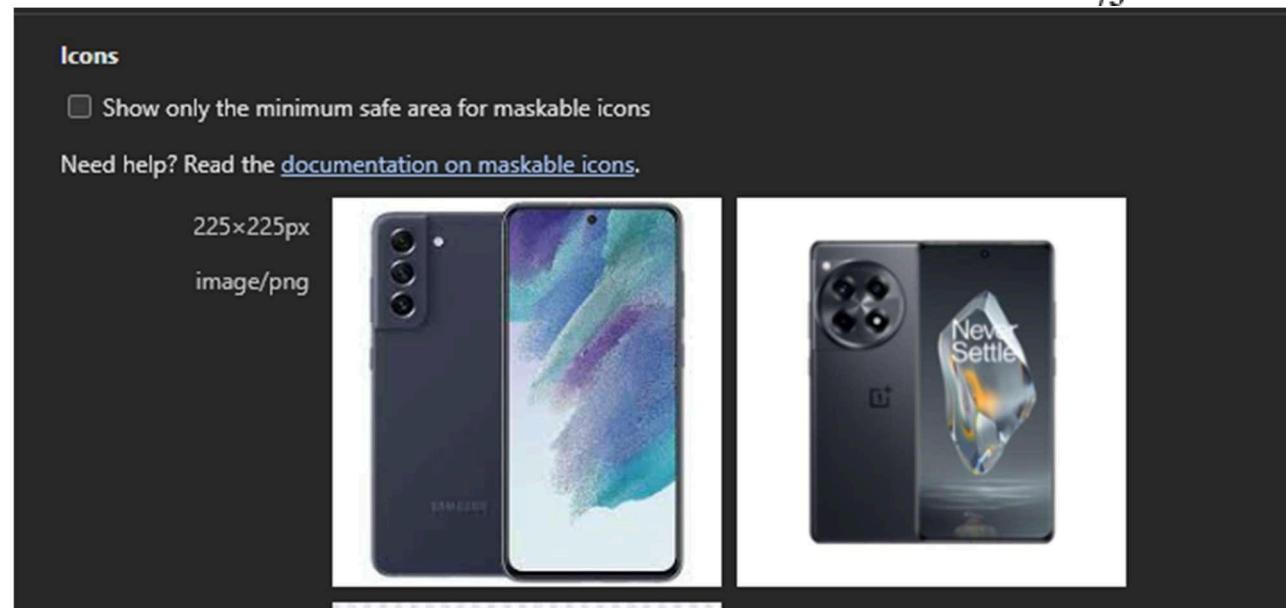


Step 5: Open the index.html file in Chrome navigate to the Application Section in the Chrome Developer Tools. Open the manifest column from the list.

The screenshot shows the Chrome DevTools Application tab open. The manifest section is visible, containing the following details:

- Identity**
 - Name: Ecommerce Homepage
 - Short name: Ecommerce
 - Description: Homepage for our Ecommerce store.
 - Computed App ID: http://127.0.0.1:5500/ (Learn more)
 - Note: id is not specified in the manifest; start_url is used instead. To specify an App ID that matches the current identity, set the id field to /.
- Presentation**
 - Start URL: /
 - Theme color: #007bff
 - Background color: #ffffff
 - Orientation
 - Display: standalone
- Protocol Handlers**
 - Define protocol handlers in the manifest to register your app as a handler for custom protocols when your app is installed.

At the bottom, there is a link: Need help? Read [URL protocol handler registration for PWAs](#).



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.

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

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

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.



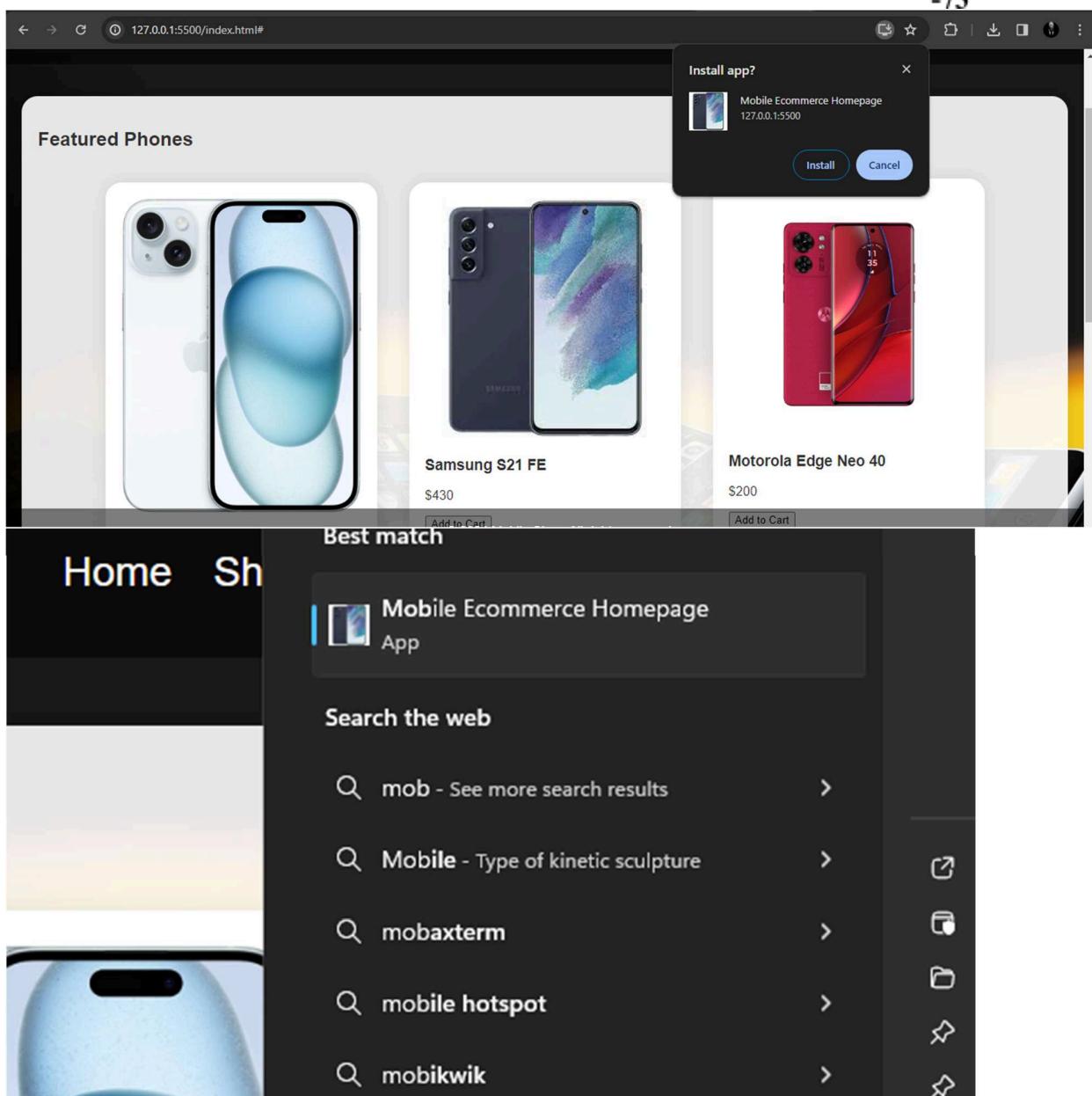
```
index.html X manifest.json ip15.jpg s24u.jpg script.js
< index.html > < html > < body > < main > < section.featured-products > < div.product-list >
  2   <html lang="en">
 10  <body>
 62    <footer>
 64      </footer>
 65
 66      <script src="script.js">
 67        window.addEventListener('load', () => {
 68          registerSW();
 69        });
 70
 71        // Register the Service Worker
 72        async function registerSW() {
 73          if ('serviceWorker' in navigator) {
 74            try {
 75              await navigator
 76                .serviceWorker
 77                  .register('serviceworker.js');
 78            } catch (e) {
 79              console.log('SW registration failed');
 80            }
 81          }
        }
```

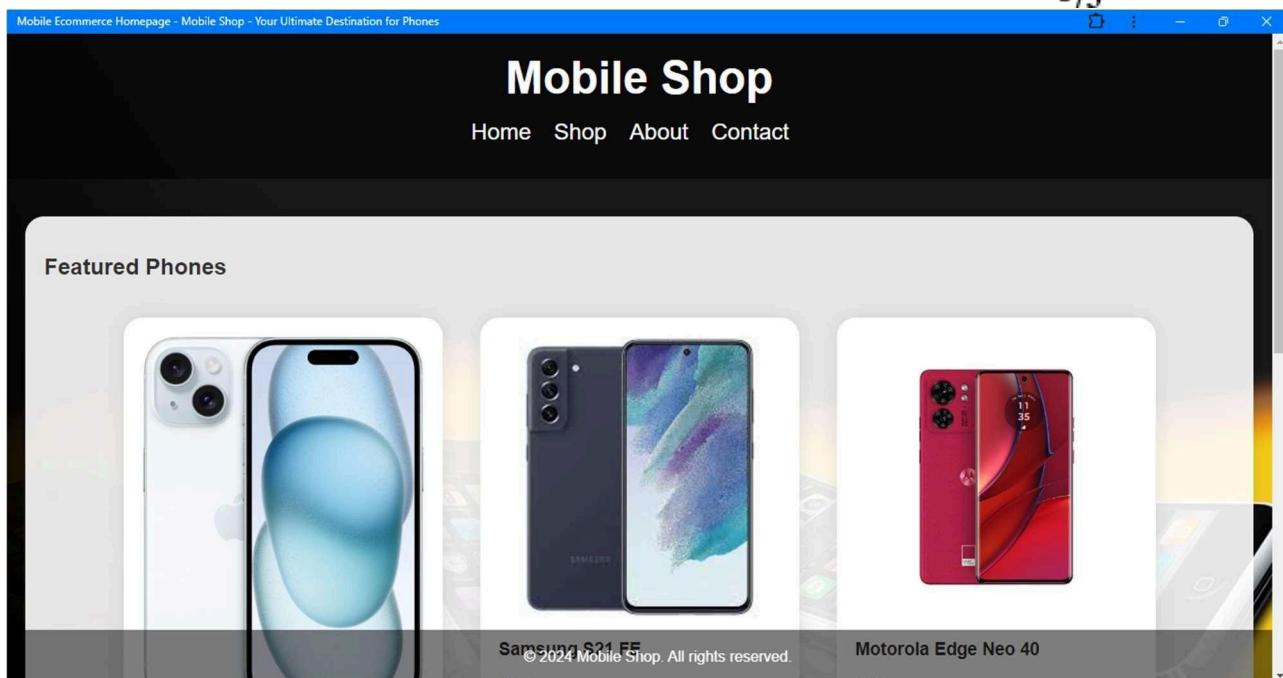
Installing the application:

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 PW