

MAD PWA Lab

EXPERIMENT NO.7

Name : Pranathi Narsupalli

Div : D15B

Roll no. : 45

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) are a type of web application that combines the best features of web and mobile apps to offer users a seamless and engaging experience. Progressive Web Apps (PWAs) represent a new paradigm in web development, seamlessly blending the strengths of web technologies with the immersive user experience of native mobile apps. By leveraging responsive design principles, service workers for offline functionality, and the Web App Manifest for installation and customization, PWAs deliver fast, reliable, and engaging experiences across devices and network conditions. With features like push notifications, home screen installation, and seamless navigation, PWAs bridge the gap between web and native apps, offering users a frictionless journey and empowering developers to build versatile, accessible, and modern web applications.

Here's a theoretical overview of PWAs:

1. Responsive Design: PWAs are designed to be responsive, meaning they can adapt and provide a great user experience across various devices, including desktops, tablets, and smartphones.

2. Progressive Enhancement: PWAs are built using progressive enhancement principles. This means they should work for all users, regardless of the browser or device they use. Advanced features are progressively enhanced for users with modern browsers and devices.

3. App-Like Experience: PWAs aim to provide an app-like experience, including smooth navigation, offline functionality, push notifications, and the ability to be added to the home screen.

4. Service Workers: One of the key technologies behind PWAs is service workers. These are scripts that run in the background and enable features like offline caching, background synchronization, and push notifications.

5. Web App Manifest: The Web App Manifest is a JSON file that contains metadata about the PWA, such as its name, icons, start URL, display mode, theme colors, and more. This manifest file is used by browsers to install the PWA and customize its appearance on the user's device.

6. Offline Functionality: PWAs can work offline or in low-connectivity environments by caching assets and data using service workers. This ensures that users can still access content and perform actions even when they are not connected to the internet.

7. Engagement Features: PWAs can engage users through features like push notifications, which allow developers to send notifications to users even when the PWA is not actively open in their browser.

8. Fast and Reliable: PWAs are designed to be fast and reliable, providing quick load times and smooth interactions to enhance the user experience.

9. Secure: PWAs are served over HTTPS to ensure data security and protect user privacy, especially when dealing with sensitive information or transactions.

10. Cross-Platform Compatibility: PWAs are platform-agnostic and can run on various operating systems and browsers, reducing development effort and reaching a broader audience.

Overall, PWAs combine the reach and accessibility of the web with the capabilities and engagement of native apps, making them a compelling choice for modern web development.

Steps :

1. **Create an HTML file for your ecommerce website that will contain a link to the manifest.json file.**

Code :

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Laptop Store</title>
  <link rel="manifest" href="manifest.json">
</head>

<body>
  <header>
    <h1>Laptop Store</h1>
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Laptops</a></li>
        <li><a href="#">About Us</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
  </header>

  <div class="container">
    <h2>Welcome to Our Laptop Store</h2>

    <div class="product-grid">
      <div class="product-item">
```

```

        
        <h3>HP</h3>
        <p>$999.99</p>
        <a href="">View Details</a>
    </div>

    <div class="product-item">
        
        <h3>Asus</h3>
        <p>$1199.99</p>
        <a href="">View Details</a>
    </div>

    <div class="product-item">
        
        <h3>Lenovo</h3>
        <p>$899.99</p>
        <a href="">View Details</a>
    </div>

</div>
</div>

<div class="container product-detail">
    
    <div class="product-info">
        <h2>HP</h2>
        <p>$999.99</p>
        <p>Description: Lorem ipsum dolor sit amet, consectetur
adipiscing elit.</p>
        <p>Specifications: Lorem ipsum dolor sit amet, consectetur
adipiscing elit.</p>
        <button>Add to Cart</button>
    </div>
</div>

<div class="container product-detail">
    
    <div class="product-info">
        <h2>Lenovo</h2>
        <p>$899.99</p>

```

```

        <p>Description: Lorem ipsum dolor sit amet, consectetur
adipiscing elit.</p>
        <p>Specifications: Lorem ipsum dolor sit amet, consectetur
adipiscing elit.</p>
        <button>Add to Cart</button>
    </div>
</div>
<div class="container product-detail">
    
    <div class="product-info">
        <h2>Asus</h2>
        <p>$1199.99</p>
        <p>Description: Lorem ipsum dolor sit amet, consectetur
adipiscing elit.</p>
        <p>Specifications: Lorem ipsum dolor sit amet, consectetur
adipiscing elit.</p>
        <button>Add to Cart</button>
    </div>
</div>
</body>
</html>

```

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

Manifest.json:

```
1  {
2    "name": "Your Ecommerce PWA",
3    "short_name": "Ecommerce PWA",
4    "description": "An ecommerce Progressive Web App for selling laptops.",
5    "start_url": "/",
6    "display": "standalone",
7    "background_color": "#ffffff",
8    "theme_color": "#333333",
9    "icons": [
10     {
11       "src": "path/to/icon-72x72.png",
12       "sizes": "72x72",
13       "type": "image/png"
14     },
15     {
16       "src": "path/to/icon-96x96.png",
17       "sizes": "96x96",
18       "type": "image/png"
19     },
20     {
21       "src": "path/to/icon-192x192.png",
22       "sizes": "192x192",
23       "type": "image/png"
24     },
25     {
26       "src": "path/to/icon-512x512.png",
27       "sizes": "512x512",
28       "type": "image/png"
29     }
30   ]
31 }
```

3. Serve the website on live server.

Laptop Store

[Home](#) [Laptops](#) [About Us](#) [Contact](#)

Welcome to Our Laptop Store



HP

\$999.99

[View Details](#)



Asus

\$1199.99

[View Details](#)



Lenovo

\$899.99

[View Details](#)



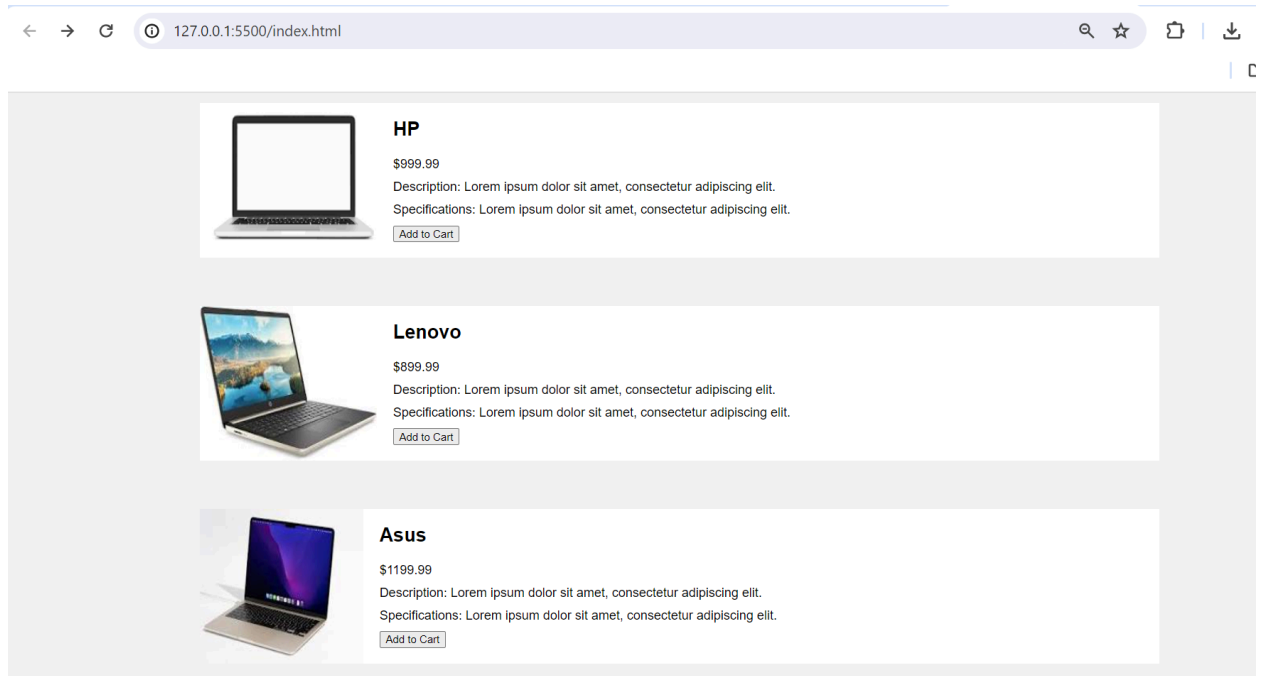
HP

\$999.99

Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

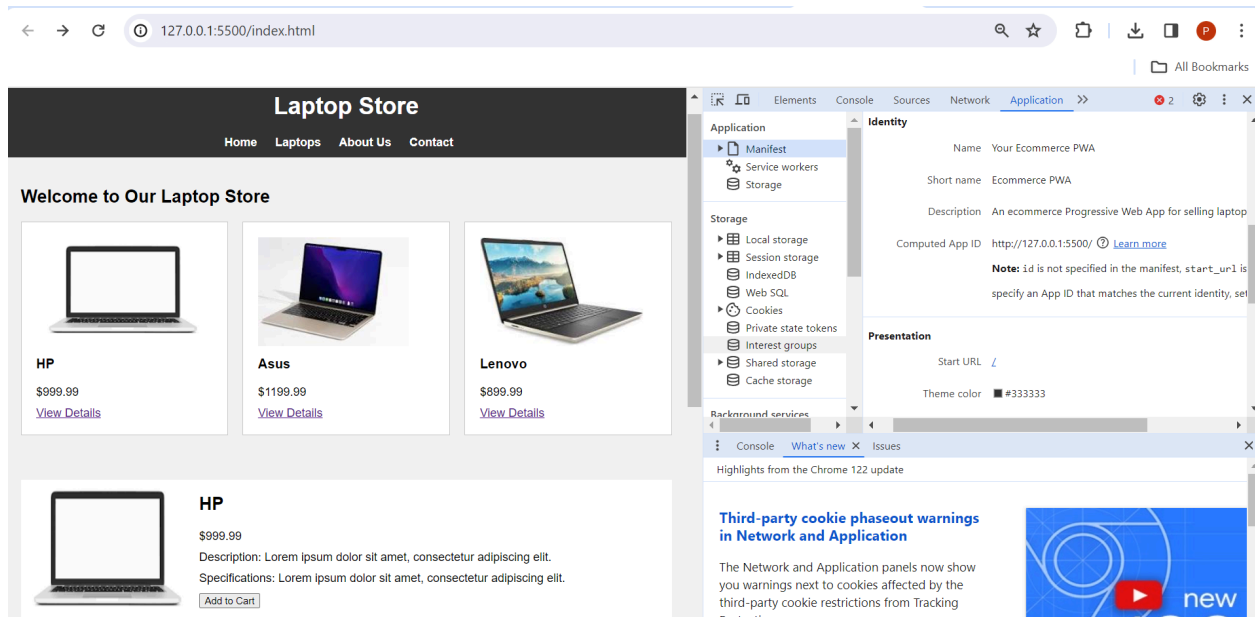
Specifications: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

[Add to Cart](#)

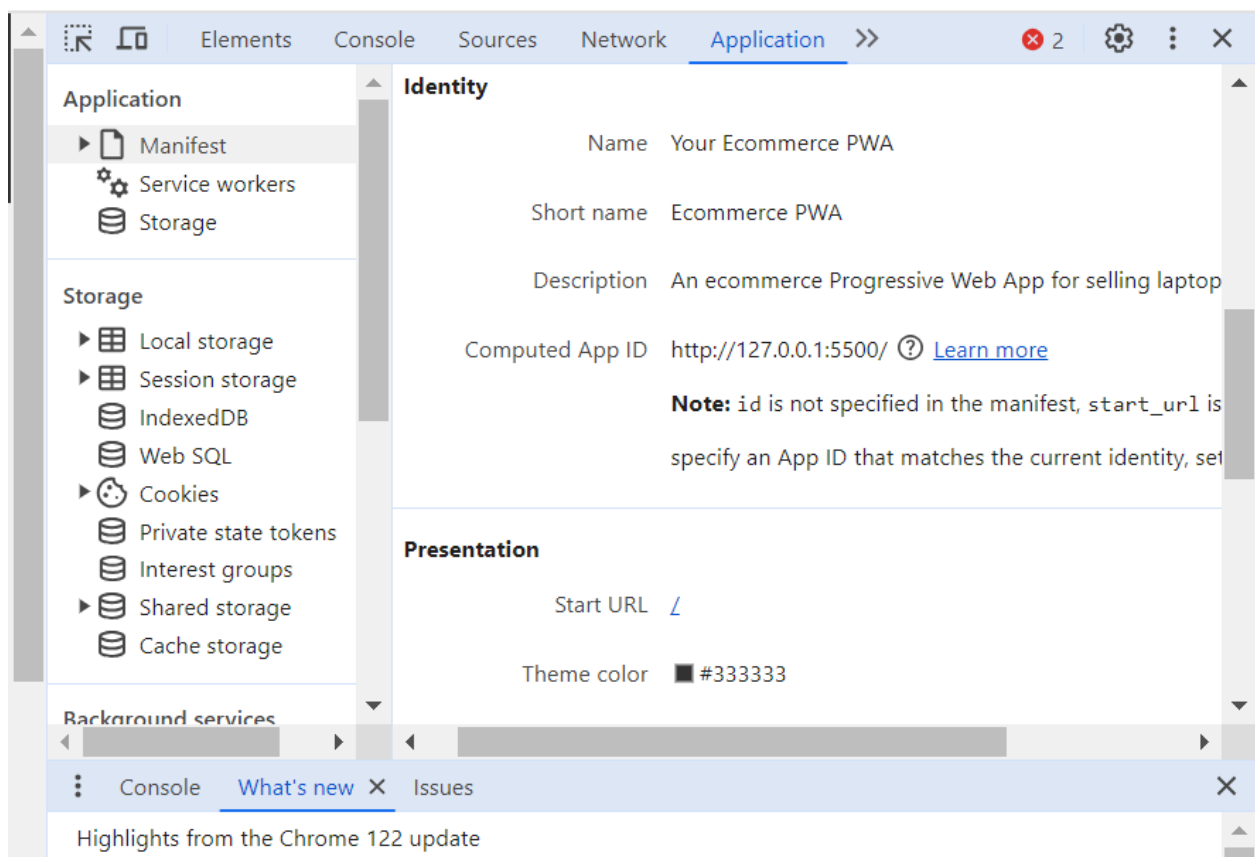


4. Create a new folder named images and place all the icons related to the application in that folder. It is recommended to have the dimensions of the icons at least 192 by 192 pixels and 512 by 512 pixels. The image name and dimensions should match that of the manifest file.

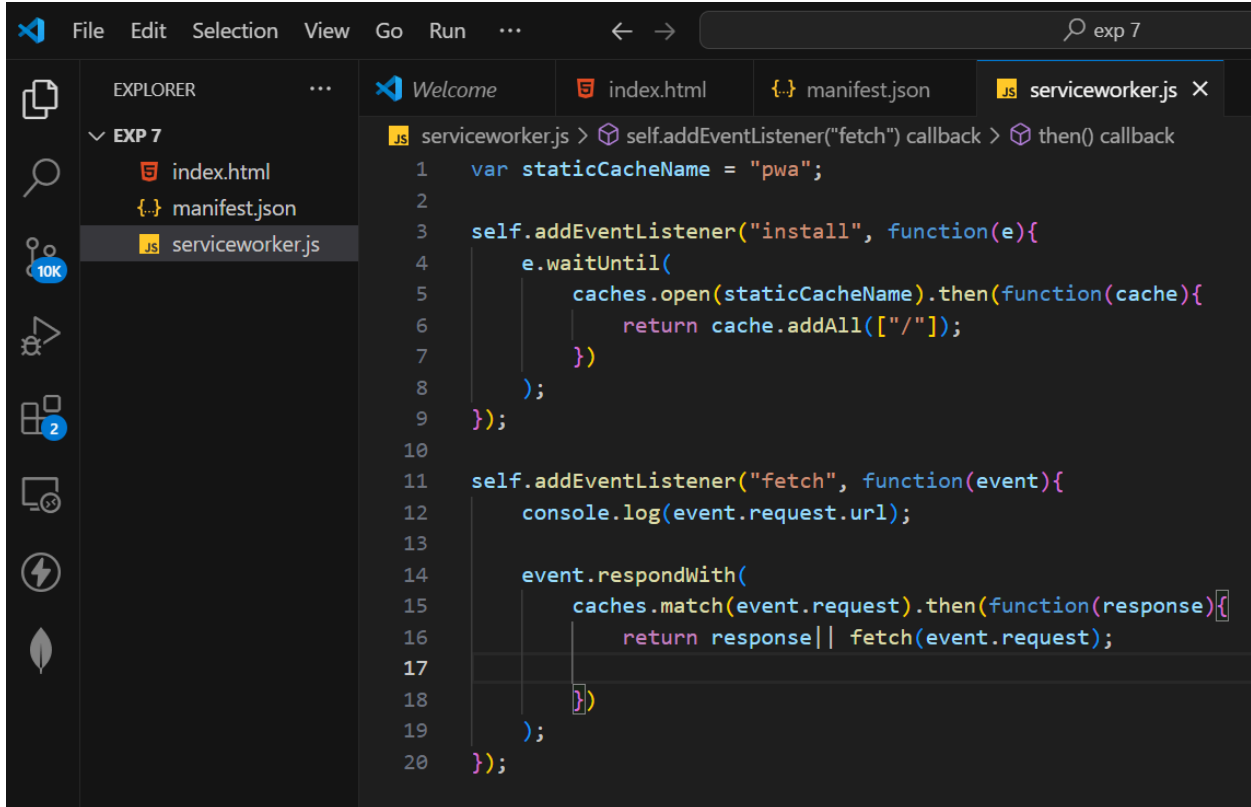
5. Serve the directory using a live server so that all files are accessible.



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

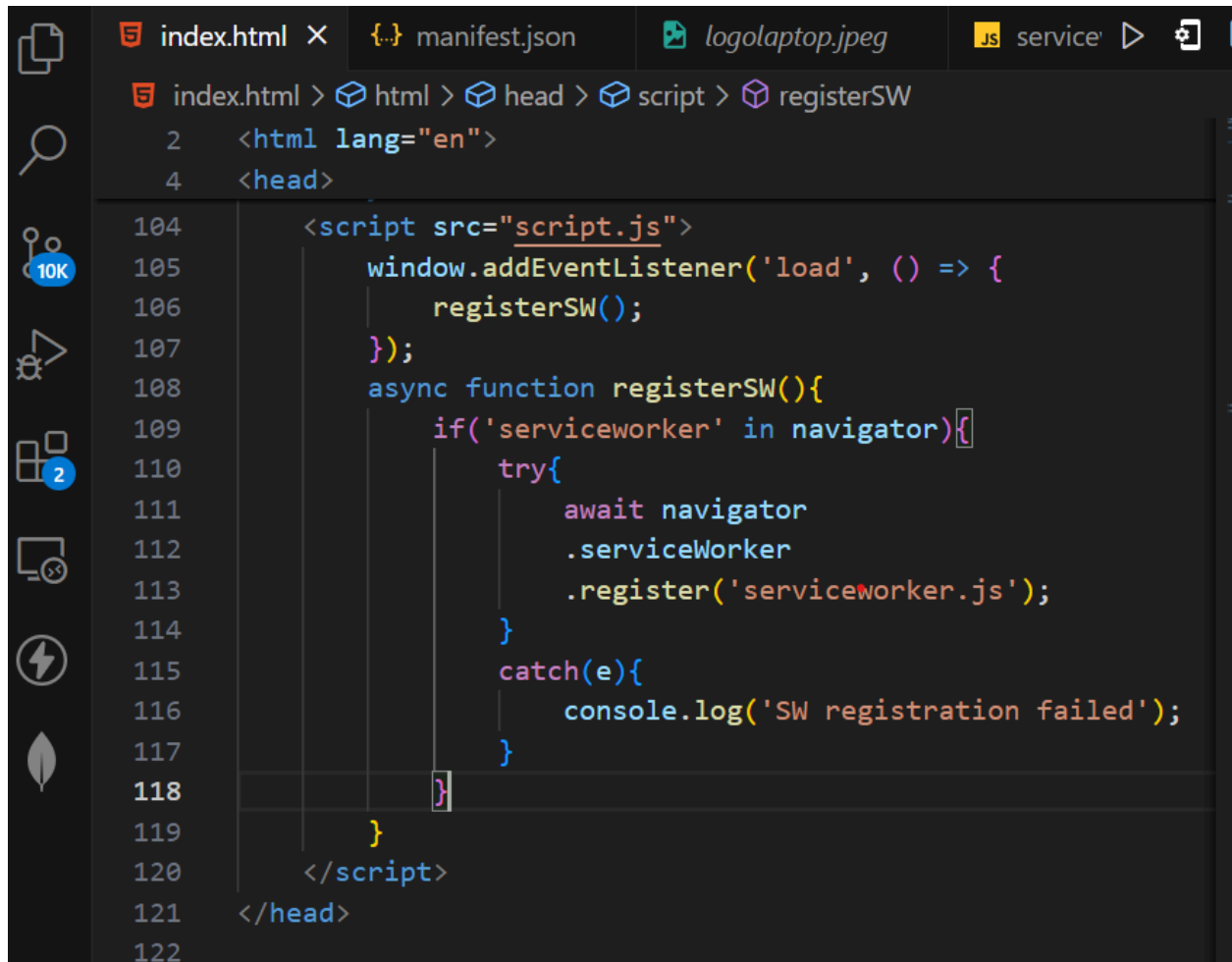


7. Under the installability tab, it would show that no service worker is detected. We will need to create another file for the PWA, that is, `serviceworker.js` in the same directory. This file handles the configuration of a service worker that will manage the working of the application.

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'EXP 7' with three files: 'index.html', 'manifest.json', and 'serviceworker.js'. The 'serviceworker.js' file is selected and its content is displayed in the main editor area. The code defines a service worker with an 'install' event listener that opens a cache named 'pwa' and adds all files to it. It also has a 'fetch' event listener that logs the request URL and responds with the cached version or fetches it from the network.

```
1  var staticCacheName = "pwa";
2
3  self.addEventListener("install", function(e){
4      e.waitUntil(
5          caches.open(staticCacheName).then(function(cache){
6              return cache.addAll(["/"]);
7          })
8      );
9  });
10
11 self.addEventListener("fetch", function(event){
12     console.log(event.request.url);
13
14     event.respondWith(
15         caches.match(event.request).then(function(response){
16             return response || fetch(event.request);
17         })
18     );
19 });
20
```

8. 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`.

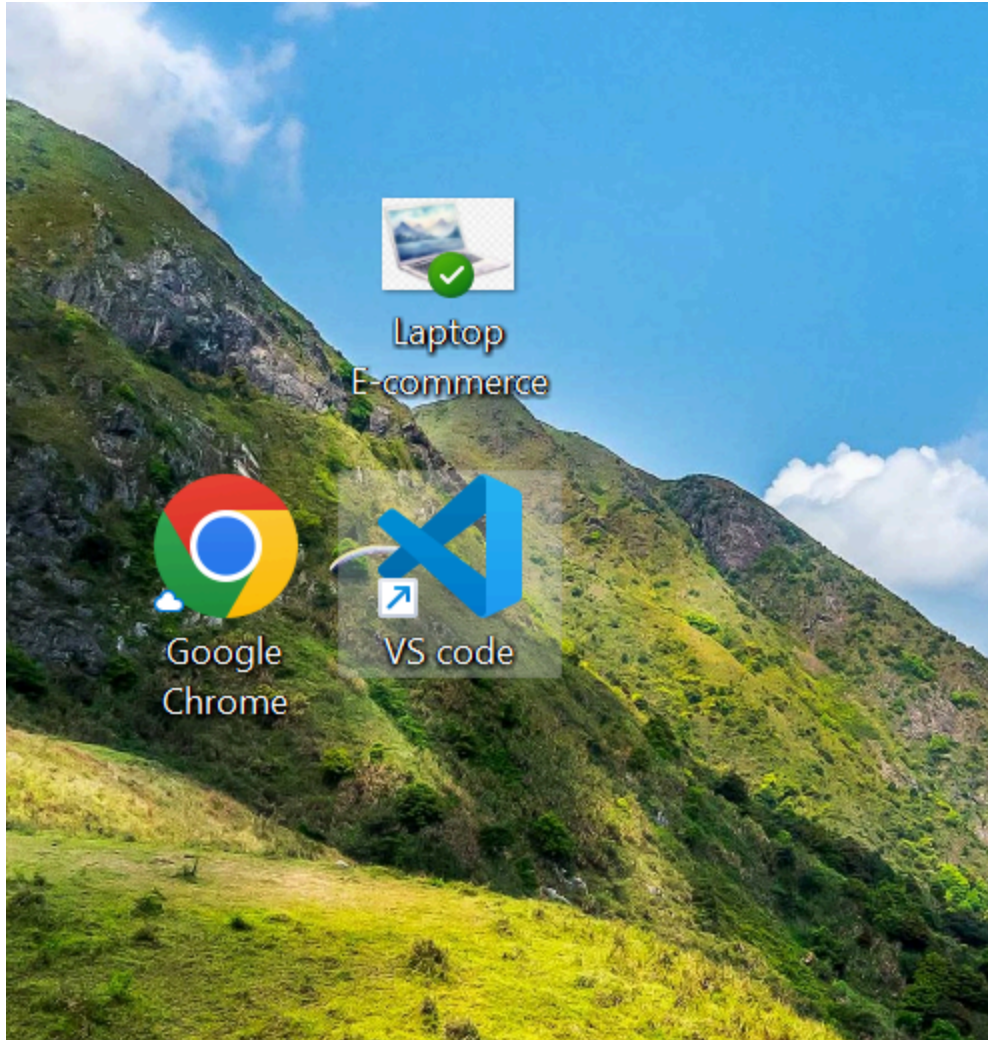


The screenshot shows a code editor with a dark theme. The top of the editor has a breadcrumb trail: `index.html > html > head > script > registerSW`. The code is as follows:

```
2 <html lang="en">
4 <head>

104 <script src="script.js">
105   window.addEventListener('load', () => {
106     registerSW();
107   });
108   async function registerSW(){
109     if('serviceworker' in navigator){
110       try{
111         await navigator
112           .serviceWorker
113           .register('serviceworker.js');
114       }
115       catch(e){
116         console.log('SW registration failed');
117       }
118     }
119   }
120 </script>
121 </head>
122
```

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.



CONCLUSION : Hence we have understood and studied the working of Progressive web applications and the features of Progressive web applications. Also, we have implemented the add to homescreen feature in Progressive web applications on our e-commerce website. 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.