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 home screen 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.

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.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>laptops</title>

    <link rel="shortcut icon" href="favicon.ico">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/al
l.min.css">
    <link rel="stylesheet" href="style.css">
    <link rel="manifest" href="manifest.json">
    <script src="script.js"></script>

</head>
<body>
    <header>
        <div class="container">
            <h1><i class="fas fa-laptop"></i> Laptops </h1>
            <nav>
                <ul>
                    <li><a href="#"><i class="fas fa-home"></i>
Home</a></li>
                    <li><a href="#"><i class="fas
fa-shopping-cart"></i> Shop</a></li>
                    <li><a href="#"><i class="fas fa-envelope"></i>
Contact</a></li>
                </ul>
            </nav>
        </div>
    </header>

    <section class="banner">
        <div class="container">
            <img src="images/heroimage.jpg" alt="Banner Image">
```

```
            <div class="overlay-text">
                <p>Discover the latest newly launched Laptops</p>
                <a href="#" class="btn">Buy Now</a>
            </div>
        </div>
    </section>

    <section class="featured-products">
        <div class="container">
            <h2>Featured Products</h2>
            <div class="product-grid">
                <div class="product">
                    <img src="images/laptop1.jpg" alt="Product 1">
                    <h3>Laptop 1</h3>
                    <p>$19.99</p>
                </div>
                <div class="product">
                    <img src="images/laptop2.jpg" alt="Product 2">
                    <h3>Laptop 2</h3>
                    <p>$24.99</p>
                </div>
                <div class="product">
                    <img src="images/laptop2.jpg" alt="Product 3">
                    <h3>Laptop 3</h3>
                    <p>$54.99</p>
                </div>
            </div>
        </div>
    </section>


    <footer>
        <div class="container">
            <p>&copy; 2024 Laptop store</p>
        </div>
    </footer>


    <script>
        window.addEventListener('load', () => {
          registerSW();
          askPermissionForPushNotification();
        });
```

```
        async function askPermissionForPushNotification() {
          if (!('Notification' in window)) {
            console.log('Notifications not supported in this browser');
            return;
          }

          try {
            const permission = await Notification.requestPermission();
            if (permission === 'granted') {
              subscribeToPushService();
            }
          } catch (error) {
            console.error('Error while requesting permission:', error);
          }
        }

        async function subscribeToPushService() {
          try {
            const registration = await navigator.serviceWorker.ready;
            const subscription = await
registration.pushManager.subscribe({
              userVisibleOnly: true,
              applicationServerKey: 'your_public_key_here'
              // replace 'your_public_key_here' with your actual public
key
            });
            console.log('Push subscription:', subscription);
            // Send subscription to server for sending push
notifications
          } catch (error) {
            console.error('Error while subscribing to push service:',
error);
          }
        }

        async function registerSW() {
          if ('service' in navigator) {
            try {
              await navigator
                    .serviceWorker
                    .register('service.js');
            }
```

```
            catch (e) {
              console.log('SW registration failed');
            }
          }
        }
    </script>

</body>
</html>
```

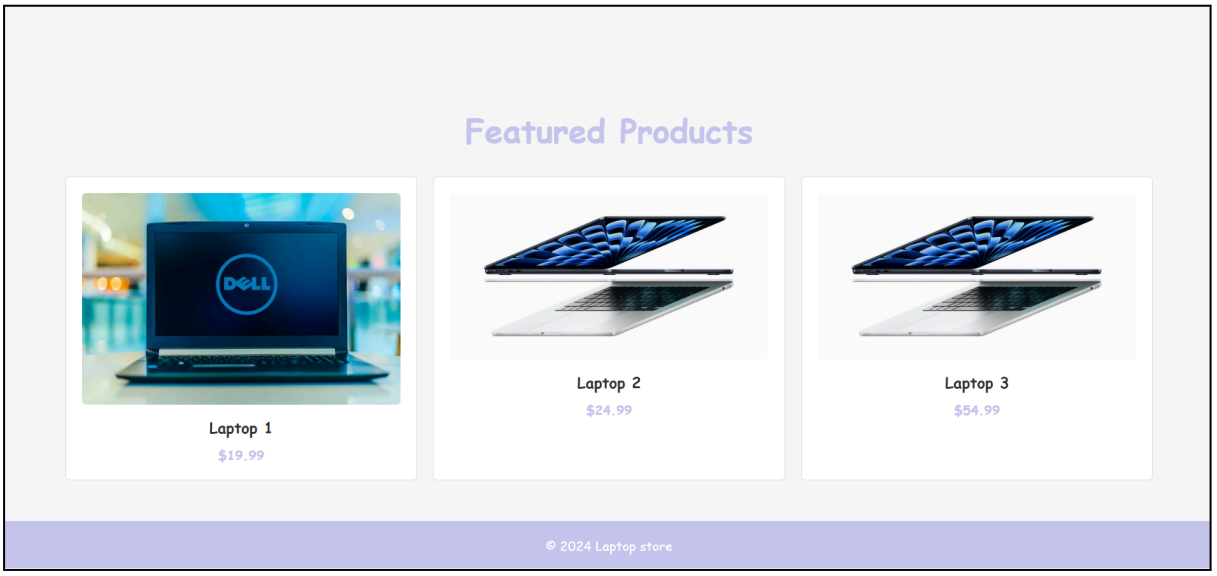
Step 2: Create a manifest.json file in the same directory. This file basically contains information about the web application. Some basic information includes the application name, starting URL, theme color, and icons. All the information required is specified in the JSON format. The source and size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

```
{
    "name": "Laptop",
    "short_name": "Laptops",
    "start_url": "laptop.html",
    "scope": "./",
    "icons": [
        {
            "src": "images/heroimage.jpg",
            "type": "image/jpg"
        },
        {
            "src": "images/laptop1.jpg",
            "sizes": "512x512",
            "type": "images/jpg"
        },
        {
            "src": "images/laptop2.jpg",
            "sizes": "512x512",
            "type": "images/jpg"
        },
        {
            "src": "images/laptop3.jpg",
            "sizes": "512x512",
            "type": "images/jpg"
        }
    ],
```

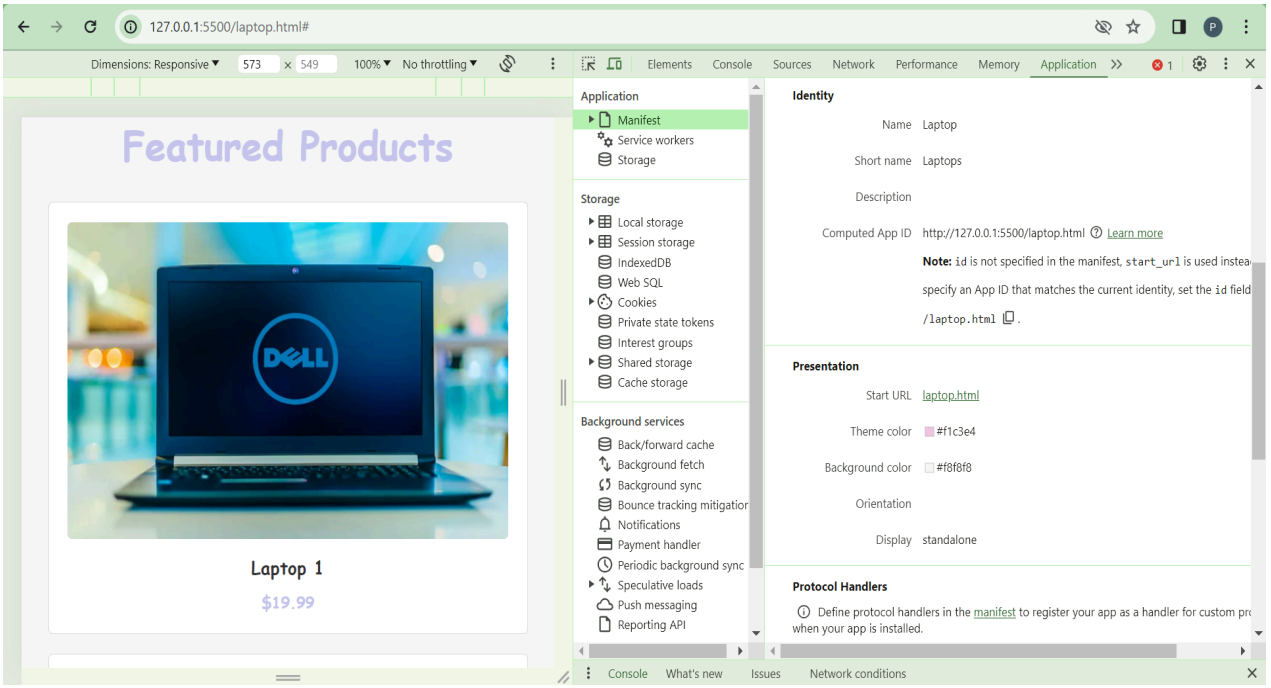
```
    "theme_color": "#f1c3e4",
    "background_color": "#f8f8f8",
    "display": "standalone"
}
```

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.

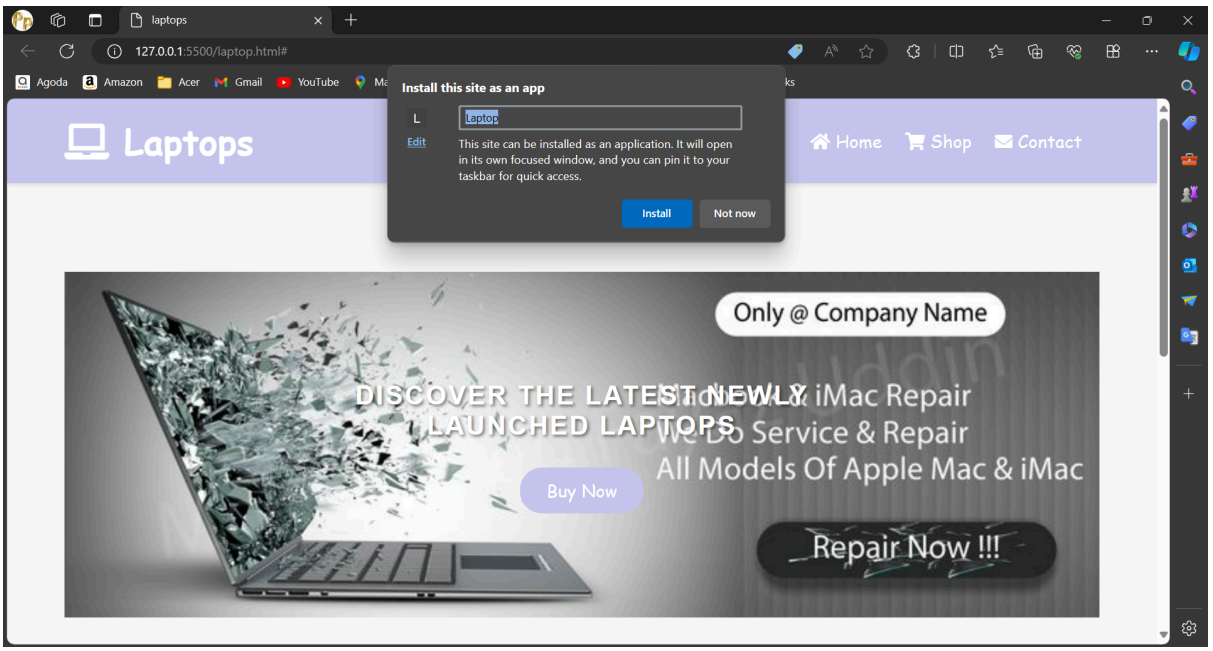
```
var CACHE_NAME = 'laptop-ecommerce-cache-v1';
```

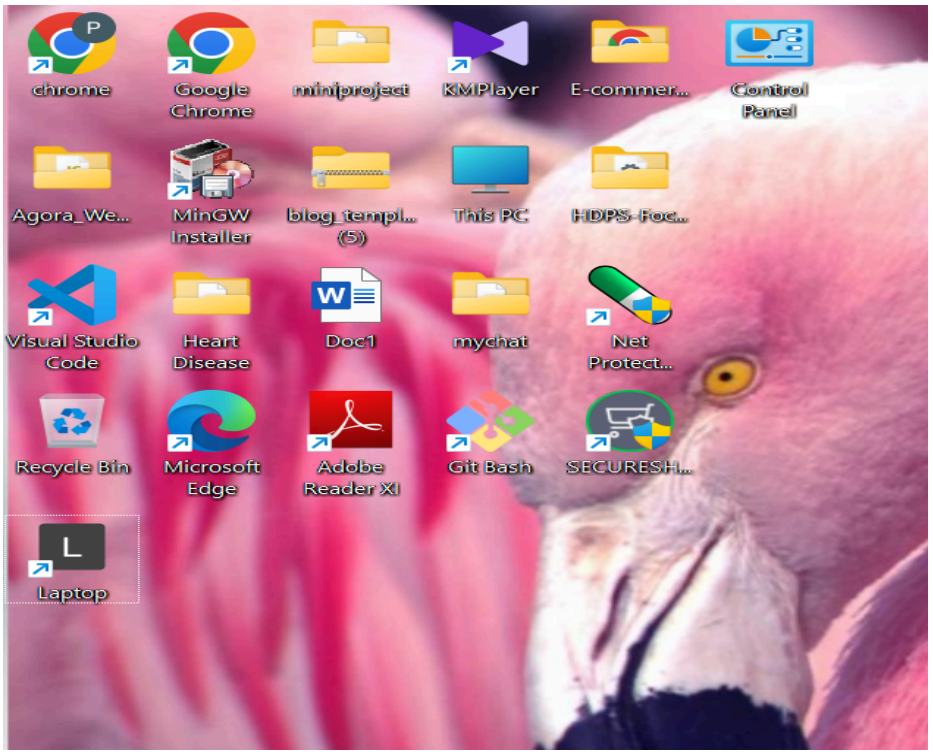
```
// Define the files to be cached
var urlsToCache = [
    '/',
    '/laptop.html',
    '/style.css',
    '/script.js',
    '/manifest.json',
    '/favicon.ico',
    '/images/heroiamge.jpg', // Add paths to images
    '/images/laptop1.jpg',
    '/images/laptop2.jpg',
    '/images/laptop3.jpg',
    // Add more files to be cached as needed
];

// Install the service worker and cache the static assets
self.addEventListener('install', function(event) {
```

```
    event.waitUntil(
        caches.open(CACHE_NAME)
            .then(function(cache) {
                console.log('Opened cache');
                return cache.addAll(urlsToCache);
            })
    );
});

// Fetch event - serve assets from cache if available, otherwise fetch
from network
self.addEventListener('fetch', function(event) {
    event.respondWith(
        caches.match(event.request)
            .then(function(response) {
                // Cache hit - return response
                if (response) {
                    return response;
                }
                // Clone the request to avoid consuming it
                var fetchRequest = event.request.clone();

                // Fetch from network
                return fetch(fetchRequest).then(
                    function(response) {
                        // Check if valid response
                        if(!response || response.status !== 200 ||
response.type !== 'basic') {
                            return response;
                        }

                        // Clone the response
                        var responseToCache = response.clone();

                        // Open cache and add the response
                        caches.open(CACHE_NAME)
                            .then(function(cache) {
                                cache.put(event.request,
responseToCache);
                            });

                        return response;
                    }
```

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

// Activate event - clean up old caches
self.addEventListener('activate', function(event) {
    event.waitUntil(
        caches.keys().then(function(cacheNames) {
            return Promise.all(
                cacheNames.filter(function(cacheName) {
                    // Delete old cache
                    return
cacheName.startsWith('laptop-ecommerce-cache-') && cacheName !==
CACHE_NAME;
                }).map(function(cacheName) {
                    return caches.delete(cacheName);
                })
            );
        })
    );
});
```

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 : In summary, it's crucial to create detailed information about your Progressive Web Application (PWA) for an online store. This includes things like the app's name, description, icons, and colors, which help users add the app to their device's home screen. By paying attention to these details in the manifest.json file, developers can greatly improve how easily users can access and enjoy their PWAs.