Experiment 7

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 web applications that use modern web technologies to provide users with a native app-like experience directly through a web browser. Here are some key theories and concepts related to PWAs:

1. Responsive Design: PWAs are built with responsive design principles in mind, ensuring that they adapt well to various screen sizes and devices. This is crucial for providing a seamless experience across desktops, tablets, and mobile phones.

2. Service Workers: One of the fundamental components of PWAs is the use of service workers. These are scripts that run in the background, enabling features like offline functionality, push notifications, and caching of assets. Service workers enhance performance and user experience by reducing reliance on network connectivity.

3. App Shell Architecture: PWAs often utilize the app shell architecture, where the basic UI elements (such as headers, navigation menus, and footer) are cached locally. This allows the app to load quickly, even in offline mode, by displaying the cached shell while fetching dynamic content.

4. Progressive Enhancement: The concept of progressive enhancement is central to PWAs. It involves starting with a basic functional version of the app that works on all devices and browsers, and then progressively enhancing it with advanced features for modern browsers that support them. This approach ensures a wider reach and backward compatibility.

5. Web App Manifest: PWAs use a web app manifest, which is a JSON file containing metadata about the app, such as its name, icons, theme colors, and display mode. This metadata is used by browsers to create an app-like experience when users add the PWA to their home screens.

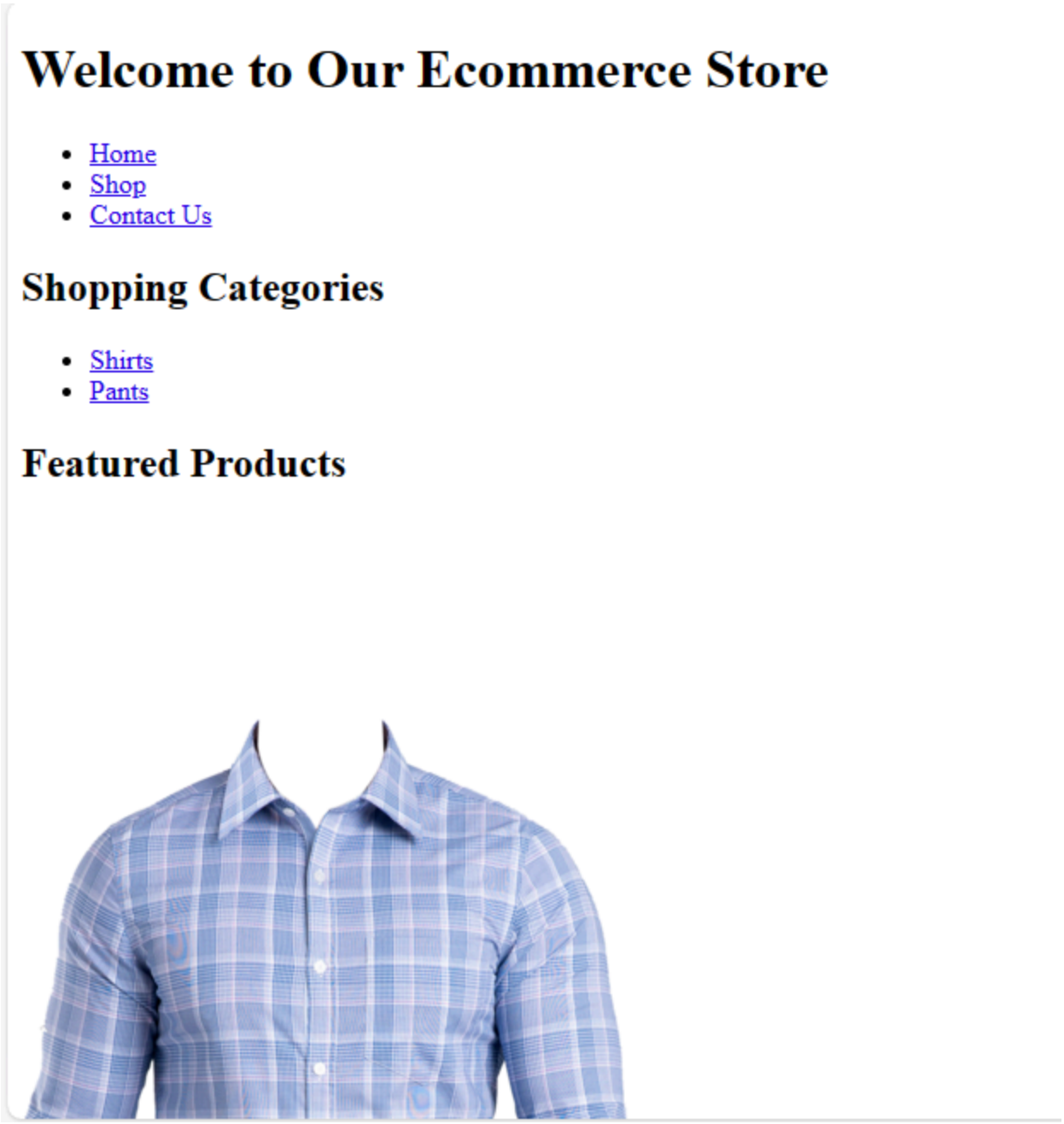
6. Offline Capabilities: PWAs can function offline or with limited network connectivity, thanks to service workers and caching strategies. They can store essential data locally and synchronize with the server when a connection is available again, providing a reliable experience regardless of network conditions.

PWAs combine the best aspects of web development and mobile app development, offering a compelling solution for delivering fast, engaging, and reliable experiences on the web.

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>Ecommerce Store</title>
    <link rel="manifest" href="manifest.json">
</head>

<body>
    <header>
        <h1>Welcome to Our Ecommerce Store</h1>
    </header>
    <nav>
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">Shop</a></li>
            <li><a href="#">Contact Us</a></li>
        </ul>
    </nav>
    <aside>
        <h2>Shopping Categories</h2>
        <ul>
            <li><a href="#">Shirts</a></li>
            <li><a href="#">Pants</a></li>
        </ul>
    </aside>
    <main>
        <section>
            <h2>Featured Products</h2>
            <div class="product">
                <img src="assets/shirt.jpg" alt="Shirt">
                <h3>Men's Shirt</h3>
                <p>$25.99</p>
                <button>Add to Cart</button>
            </div>
            <div class="product">
                <img src="assets/pant.jpg" alt="Pant">
                <h3>Women's Pant</h3>
```

```
            <p>$29.99</p>
            <button>Add to Cart</button>
          </div>
        </section>
    </main>
    <footer>
        <p>&copy; 2024 Ecommerce Store. All rights reserved.</p>
    </footer>
</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": "Ecommerce Store",
  "short_name": "Store",
```

```
    "start_url": "/index.html",
    "display": "standalone",
    "theme_color": "#ffffff",
    "background_color": "#f0f0f0",
    "icons": [
        {
            "src": "icon.png",
            "sizes": "192x192",
            "type": "image/png"
        },
        {
            "src": "icon512.png",
            "sizes": "512x512",
            "type": "image/png"
        }
    ]
}
```

Step 3: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.

ServiceWorker.js

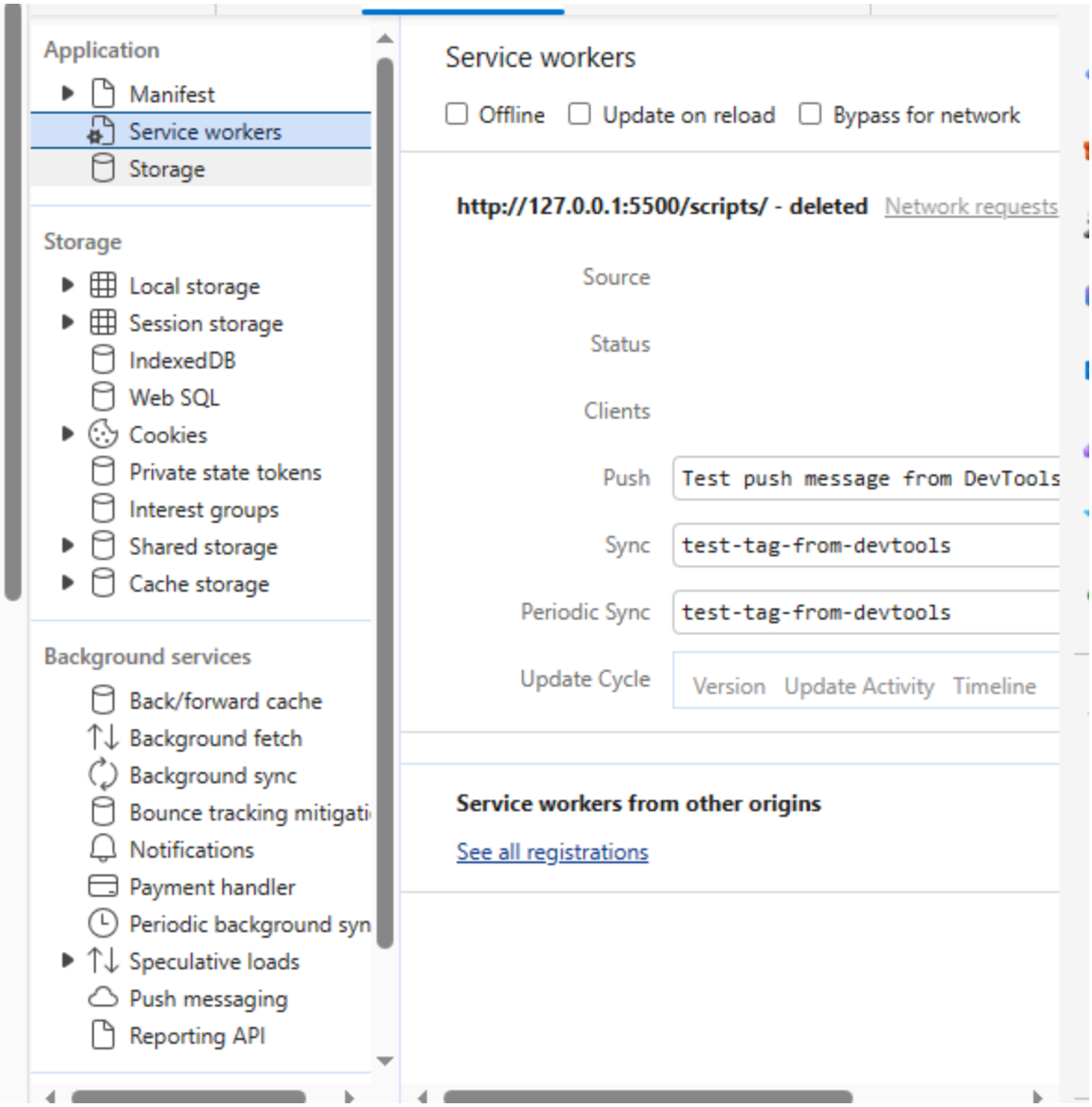
```
const CACHE_NAME = 'ecommerce-store-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/manifest.json',
  '/icon.png',
  '/icon512.png',
  // Add more URLs to cache as needed for your application
];

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Cache opened');
        return cache.addAll(urlsToCache);
      })
  );
});

self.addEventListener('fetch', event => {
```

```
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        if (response) {
          return response;
        }
        return fetch(event.request);
      })
  );
});
```

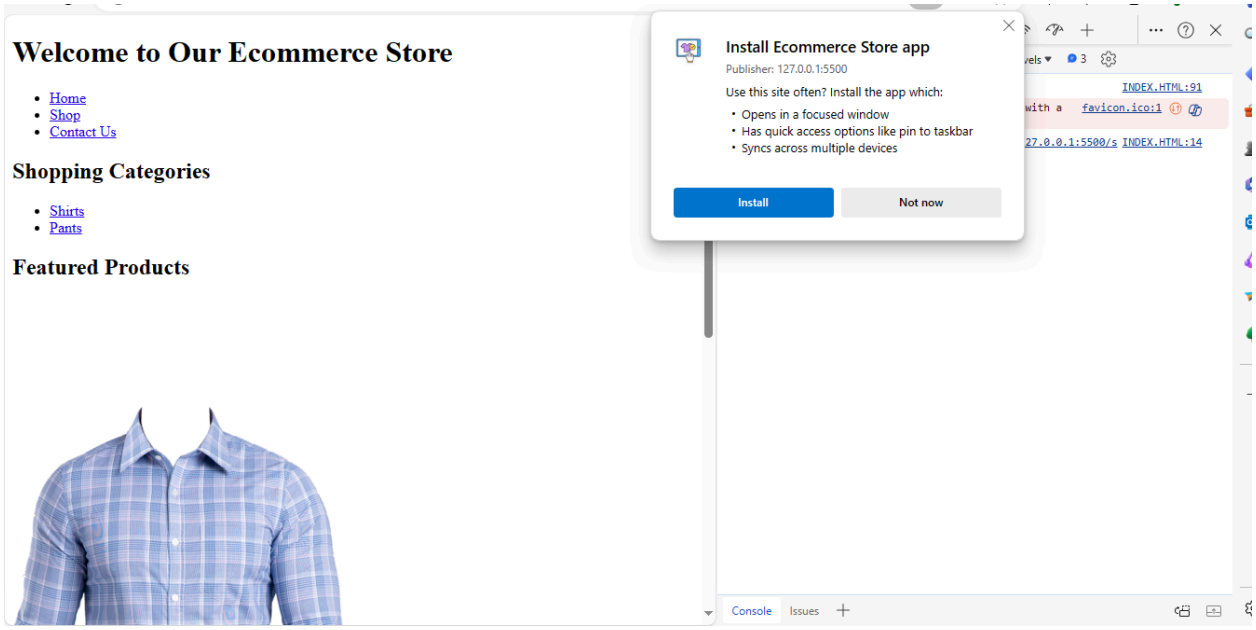
Step 4: The next step is to link the service worker file to index.html. This is done by adding a short JavaScript script to the index.html created in the above steps. Add the below code inside the script tag in index.html.

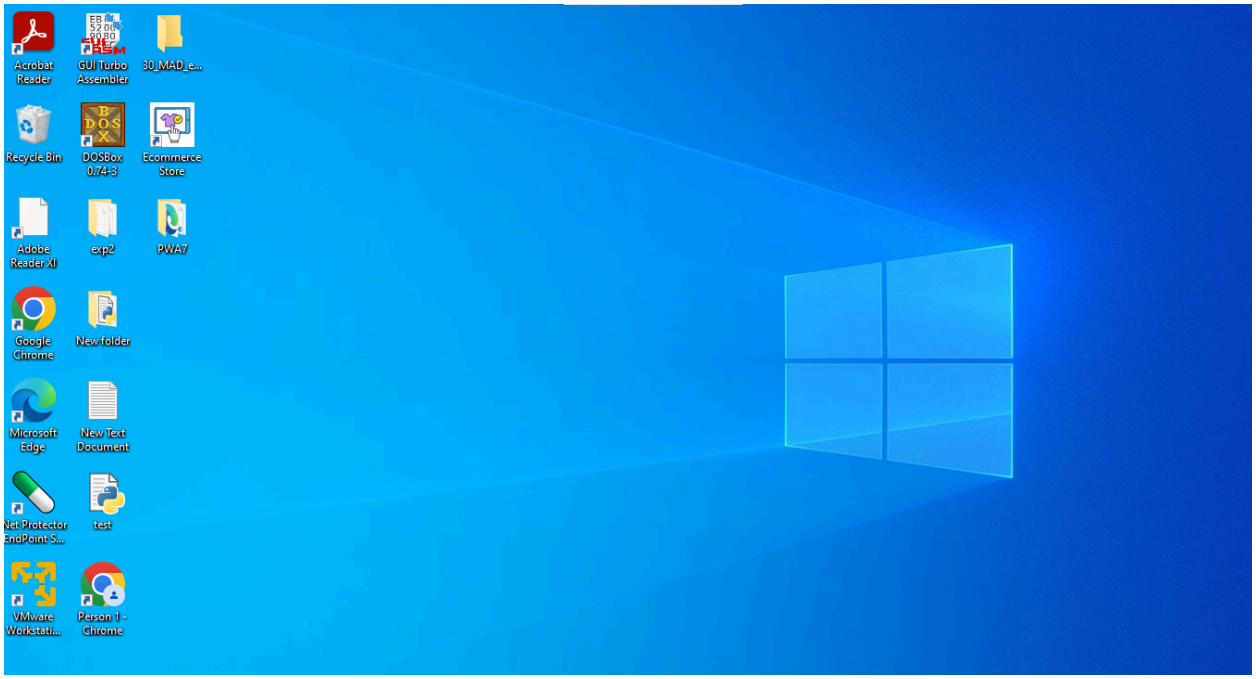
```
<script>
  if ('serviceWorker' in navigator) {
    window.addEventListener('load', () => {
      navigator.serviceWorker.register('/serviceworker.js')
        .then(registration => {
          console.log('Service Worker registered with scope:', registration.scope);
        })
        .catch(error => {
          console.error('Service Worker registration failed:', error);
        });
    });
  }
</script>
```

Step 5: Serve the directory using a live server so that all files are accessible. Open the index.html file in Chrome navigate to the Application Section in the Chrome Developer Tools. Open the manifest column from the list. Under the installability tab, it would show that service worker.



Step 6: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.