Experiment No: 10

AIM: To study and implement deployment of Ecommerce PWA to GitHub Pages

THEORY:

Add to Home screen

Add to Home screen (or A2HS for short) is a feature available in modern browsers that allows a user to "install" a web app, ie. add a shortcut to their Home screen representing their favorite web app (or site) so they can subsequently access it with a single tap. This guide explains how A2HS is used, and what you need to do as a developer to allow your users to take advantage of it.

To enable your app to be added to a **Home screen**, it needs the following:

- To be served over HTTPs — the web is increasingly being moved in a more secure direction, and many modern web technologies (A2HS included) will work only on secure contexts.
- To have a manifest file with the correct fields filled in, linked from the HTML head.
- To have an appropriate icon available for displaying on the Home screen.
- Chrome additionally requires the app to have a service worker registered (e.g., so it can function when offline).

Manifest to make the app installable:

The manifest for our sample app looks like this:

```
{
   "short_name": "SIES-PWA",
   "name": "PWA Lab Manual",
   "icons": [
      {
       "src": "assets/icons/icon-48x48.png",
       "sizes": "48x48",
       "type": "image/png",
       "purpose": "any"
       },
       {
       "src": "assets/icons/icon-72x72.png",
       "sizes": "72x72",
       "type": "image/png",
       "purpose": "any"
       },
       {
       "src": "assets/icons/icon-96x96.png",
       "sizes": "96x96",
       "type": "image/png",
       "purpose": "any"
```

```
      },
      {
       "src": "assets/icons/icon-128x128.png",
       "sizes": "128x128",
       "type": "image/png",
       "purpose": "any"
      },
      {
       "src": "assets/icons/icon-144x144.png",
       "sizes": "144x144",
       "type": "image/png",
       "purpose": "any"
      }
 ],
"id": "/?source=pwa",
"start_url": "/?source=pwa",
"background_color": "#3367D6",
"display": "standalone",
"scope": "/",
"theme_color": "#3367D6",
"shortcuts": [
  {
    "name": "PWA Lab Manual 1",
    "short_name": "PWA1",
    "description": "PWA Lab 1",
    "url": "?source=pwa",
    "icons": [{ "src": "assets/icons/icon-144x144.png",
        "sizes": "144x144" }]
  },
  {
    "name": "PWA Lab Manual 2",
    "short_name": "PWA2",
    "description": "PWA Lab 2",
    "url": "?source=pwa",
    "icons": [{ "src": "assets/icons/icon-144x144.png",
        "sizes": "144x144" }]
  }
],
"description": "PWA Lab ",
"screenshots": [
  {
    "src": "assets/icons/icon-512x512.png",
    "sizes": "512x512",
    "type": "image/png"
  }   ] }
```

As shown in the above manifest listing, we are including a 192 x 192 px icon for use in our app. You can include more sizes if you want; Android will choose the most appropriate size for each different use case. The type member in each icon's object specifies the icon's mimetype, so the browser can quickly read what type the icon is, and then ignore it and move to a different icon if it doesn't support it.

Link the HTML to the manifest

To finish setting up your manifest, you need to reference it from the HTML of your application's home page:

```
<link rel="manifest" href="manifest.webmanifest">
<head>
         <meta charset="UTF-8">
          <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta name="viewport" content="width=device-width, initial-scale=1.0">
           <title>PWA Demo</title>
         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
         rel="stylesheet"
         integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
         crossorigin="anonymous">
             <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
                 integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
                 crossorigin="anonymous"></script>
             <link rel="manifest" href="manifest.webmanifest">
              <script src="sw.js"></script>
         </head>
```

Refer https://github.com/bushsk/bushsk.github.io/blob/main/manifest.webmanifest

Service worker to make your site available offline:

The `sw.js` file is the ServiceWorker that defines which of the files in the application should become available offline.

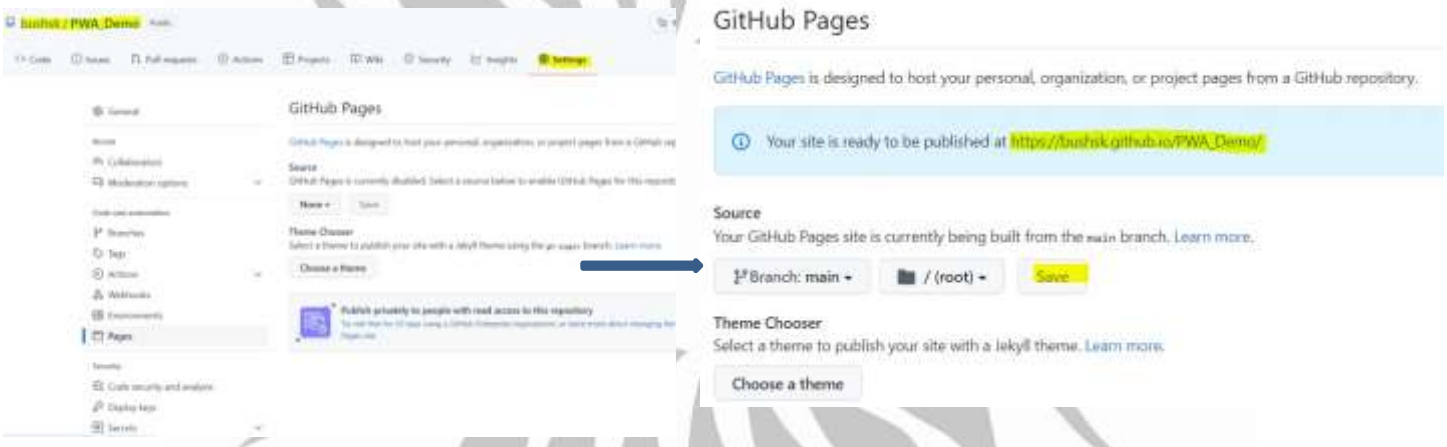
```
if ('serviceWorker' in navigator) {
        window.addEventListener('load', function () {
            navigator.serviceWorker.register('/sw.js', { scope: "/" }).then(function (registration) {
                // Registration was successful
                console.log('ServiceWorker registration successful with scope: ', registration.scope);
            }, function (err) {
                // registration failed :(
                console.log('ServiceWorker registration failed: ', err);
            });
        })
    }
// The version of the cache. Every time you change any of the files you need to change
this version (version_01, version_02…). If you don't change the version, the service
worker will give your users the old files!
var CACHE_NAME = 'my-site-cache-v1';
```

```
// The files to make available for offline use. make sure to add others to this list
  var urlsToCache = [
    '/',
    'index.html',     ];
```

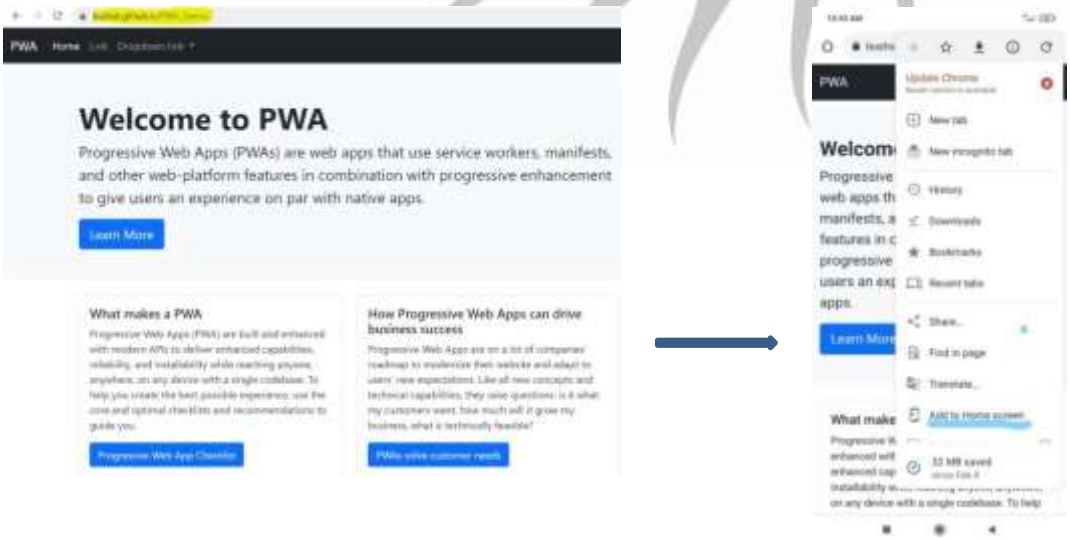
Refer https://github.com/bushsk/bushsk.github.io/blob/main/sw.js

Step 1: Push your project on to GitHub.

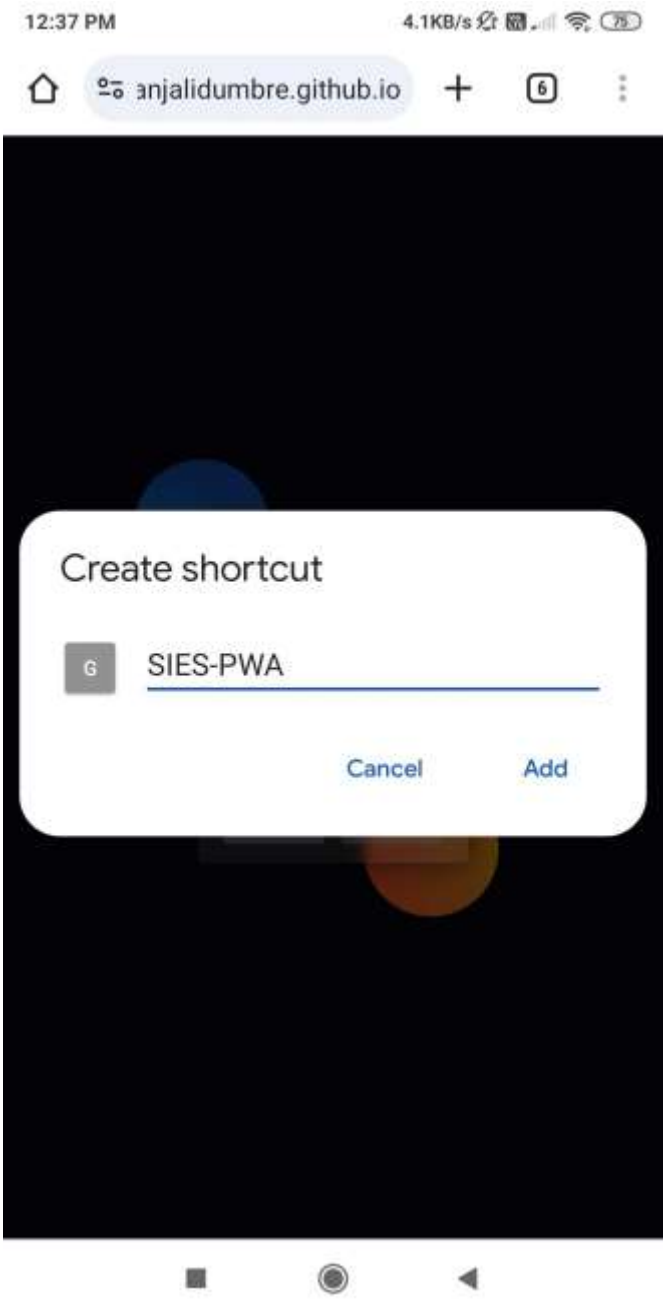
Step 2: In the **Settings** of this repository `PWA_Demo `, choose to publish the `**main**` branch as a GitHub page as shown in the following screenshot.



This means that this app is now available at https://bushsk.github.io/ . Every time we publish to the `main` branch, it triggers an action and the page is generated.




12:37 PM
1.9KB/s
anjalidumbre.github.io
Login Here
Username
Password
Log in


12:37 PM
4.1KB/s
anjalidumbre.github.io
Create shortcut
G
SIES-PWA
Cancel
Add


12:37 PM
0.0KB/s
G
SIES-PWA

Note: After A2HS and launching your app, if your webpage / website doesn't appear, then rename your Github repository name to as <username.github.io> and pull down to refresh.

A2HS on desktop

While originally intended to improve user experience on mobile OSes, there is movement to make PWAs installable on desktop platforms too.

Adding an install button :

To make our PWA installable on desktop, we first added a button to our document to allow users to do the installation — this isn't made available automatically on desktop apps, and the installation needs to be triggered by a user gesture:

```
<button class="add-button">Add to home screen</button>
```

We then gave it some simple styling:

```
.add-button {
  position: absolute;
  top: 1px;
  left: 1px;
}
```

JavaScript for handling the install

At the bottom of our index.js file, we added some JavaScript to handle the installation. First of all, we declare a deferredPrompt variable (which we'll explain later on), get a reference to our install button, and set it to display: none initially:

```
let deferredPrompt;
const addBtn = document.querySelector('.add-button');
addBtn.style.display = 'none';
```

We hide the button initially because the PWA will not be available for install until it follows the A2HS criteria. When this happens, supporting browsers will fire a beforeinstallprompt event. We can then use a handler like the one below to handle the installation:

```
window.addEventListener('beforeinstallprompt', (e) => {
  // Prevent Chrome 67 and earlier from automatically showing the prompt
  e.preventDefault();
  // Stash the event so it can be triggered later.
  deferredPrompt = e;
  // Update UI to notify the user they can add to home screen
  addBtn.style.display = 'block';
```

```
 addBtn.addEventListener('click', (e) => {
  // hide our user interface that shows our A2HS button
  addBtn.style.display = 'none';
  // Show the prompt
  deferredPrompt.prompt();
  // Wait for the user to respond to the prompt
  deferredPrompt.userChoice.then((choiceResult) => {
    if (choiceResult.outcome === 'accepted') {
      console.log('User accepted the A2HS prompt');
    } else {
      console.log('User dismissed the A2HS prompt');
    }
    deferredPrompt = null;
   });
 });
});
```

So here we:

Call Event.preventDefault() to stop Chrome 67 and earlier from calling the install prompt automatically (this behavior changed in Chrome 68).

Store the event object in the deferredPrompt variable so it can be used later on to perform the actual installation.

Set the button to display: block so it appears in the UI for the user to click.

Set a click handler for the button.

The click handler contains the following steps:

- Hide the button again with display: none — it is no longer needed once the app is installed.
- Use the prompt() method available on the beforeinstallprompt event object (stored in deferredPrompt) to trigger showing the install prompt.
- Respond to the user's interaction with the prompt using the userChoice property, again available on the beforeinstallprompt event object.
- Set deferredPrompt to null since it is no longer needed.

So when the button is clicked, the install prompt appears.

If the user selects Install, the app is installed (available as standalone desktop app), and the Install button no longer shows (the onbeforeinstallprompt event no longer fires if the app is already installed).

Conclusion: - Hence, we successfully implement deployment of Ecommerce PWA to GitHub Pages.