MAD & PWA Lab
Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

AIM : To write meta data of your PWA in a Web app manifest file to enable "add to homescreen feature".

Theory :

Online Reference:
https://developer.mozilla.org/en-US/docs/Web/Manifest
https://www.geeksforgeeks.org/making-a-simple-pwa-under-5-minutes/

Regular Web App

A regular web app is a website that is designed to be accessible on all mobile devices such that the content gets fit as per the device screen. It is designed using a web technology stack (HTML, CSS, JavaScript, Ruby, etc.) and operates via a browser. They offer various native-device features and functionalities. However, it entirely depends on the browser the user is using. In other words, it might be possible that you can access a native-device feature on Chrome but not on Safari or Mozilla Firefox because the browsers are incompatible with that feature.

Progressive Web App

Progressive Web App (PWA) is a regular web app, but some extras enable it to deliver an excellent user experience. It is a perfect blend of desktop and mobile application experience to give both platforms to the end-users.

Difference between PWAs vs. Regular Web Apps:

A Progressive Web is different and better than a Regular Web app with features like:

1. Native Experience

Though a PWA runs on web technologies (HTML, CSS, JavaScript) like a Regular web app, it gives user experience like a native mobile application. It can use most native device features, including push notifications, without relying on the browser or any other entity. It offers a seamless and integrated user experience that it is quite tough for one to differentiate between a PWA and a Native application by considering its look and feel.

2. Ease of Access

Unlike other mobile apps, PWAs do not demand longer download time and make memory space available for installing the applications. The PWAs can be shared and installed by a link, which cuts down the number of steps to install and use. These applications can easily keep an app icon on the user's home screen, making the app easily accessible to the users and helps the brands remain in the users' minds, and improving the chances of interaction.

3. Faster Services

PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

4. Engaging Approach
As already shared, the PWAs can employ push notifications and other native device features more efficiently. Their interaction does not depend on the browser user uses. This eventually improves the chances of notifying the user regarding your services, offers, and other options related to your brand and keeping them hooked to your brand. In simpler words, PWAs let you maintain the user engagement and retention rate.

5. Updated Real-Time Data Access
Another plus point of PWAs is that these apps get updated on their own. They do not demand the end-users to go to the App Store or other such platforms to download the update and wait until installed. In this app type, the web app developers can push the live update from the server, which reaches the apps residing on the user's devices automatically. Therefore, it is easier for the mobile app developer to provide the best of the updated functionalities and services to the end-users without forcing them to update their app.

6. Discoverable
PWAs reside in web browsers. This implies higher chances of optimizing them as per the Search Engine Optimization (SEO) criteria and improving the Google rankings like that in websites and other web apps.

7. Lower Development Cost
Progressive web apps can be installed on the user device like a native device, but it does not demand submission on an App Store. This makes it far more cost-effective than native mobile applications while offering the same set of functionalities.

Pros and cons of the Progressive Web App
The main features are:

Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.

Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.

App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.

Updated — Information is always up-to-date thanks to the data update process offered by service workers.

Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.

Searchable — They are identified as “applications” and are indexed by search engines.

Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
Installable — They allow the user to "save" the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.

Linkable — Easily shared via URL without complex installations.

Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two particularities: first of all the 'skeleton' of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Weaknesses refer to:

IOS support from version 11.3 onwards;
Greater use of the device battery;
Not all devices support the full range of PWA features (same speech for iOS and Android operating systems);
It is not possible to establish a strong re-engagement for iOS users (URL scheme, standard web notifications);
Support for offline execution is however limited;
Lack of presence on the stores (there is no possibility to acquire traffic from that channel);
There is no "body" of control (like the stores) and an approval process;
Limited access to some hardware components of the devices;
Little flexibility regarding "special" content for users (eg loyalty programs, loyalty, etc.).

Steps :

1. Create manifest.json file

Create a file named manifest.json in your project root or public/ folder.

```
{
 "name": "My Awesome App",
 "short_name": "AwesomeApp",
 "description": "An awesome PWA that does cool things!",
 "start_url": "/",
 "display": "standalone",
 "background_color": "#ffffff",
 "theme_color": "#1976d2",
 "icons": [
  {
   "src": "/icons/icon-192x192.png",
   "sizes": "192x192",
   "type": "image/png"
```

```
  },
  {
   "src": "/icons/icon-512x512.png",
   "sizes": "512x512",
   "type": "image/png"
  }
 ]
}
```

2. Link the manifest file in your HTML

In your main HTML file (index.html), add the following inside the <head> tag:

```
<link rel="manifest" href="/manifest.json">
<meta name="theme-color" content="#1976d2">
```

3. Add icons

Place your app icons in the specified path (e.g., /icons/icon-192x192.png and /icons/icon-512x512.png).

Make sure they meet these requirements:

- PNG format
- Sizes: 192x192 and 512x512 at least

4. Serve over HTTPS

PWA features like "Add to Home Screen" require HTTPS (except on localhost for development).

5. Register a service worker (mandatory for PWA)

In your JS file (e.g., main.js or index.js): (Here I stored in index.html)

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/service-worker.js')
      .then(reg => console.log('Service Worker registered', reg))
      .catch(err => console.log('Service Worker registration failed', err));
  });
}
```

6. Add a minimal service-worker.js

At the root level:

```
self.addEventListener('install', event => {
  console.log('Service Worker installed');
});

self.addEventListener('fetch', event => {
  // You can cache requests here
});
```

Done! Now your app will prompt “Add to Home Screen” when:

- It has a manifest.json
- A service worker is registered
- Served over HTTPS
- Visited more than once by the use

Code :

manifest.json:-

```
{
   "name": "VESIT college",
   "short_name": "VESIT",
   "start_url": "/",
   "display": "standalone",
   "background_color": "#5900b3",
   "theme_color": "black",
   "scope": ".",
   "description": "This is a PWA tutorial.",
   "icons": [
     {
       "src": "/vesitlogo.jpg",
       "sizes": "192x192",
       "type": "image/jpg"
     },
     {
       "src": "/google_logo.png",
       "sizes": "512x512",
       "type": "image/png"
     }
   ],
   "screenshots": [
  {
   "src": "/screenshot1.png",
```

```
    "type": "image/png",
    "sizes": "1280x720",
    "form_factor": "wide"
  },
  {
    "src": "/screenshot2.png",
    "type": "image/png",
    "sizes": "720x1280",
    "form_factor": "narrow"
  }
]

  }
```

Add the link tag to link to the manifest.json file

index.html

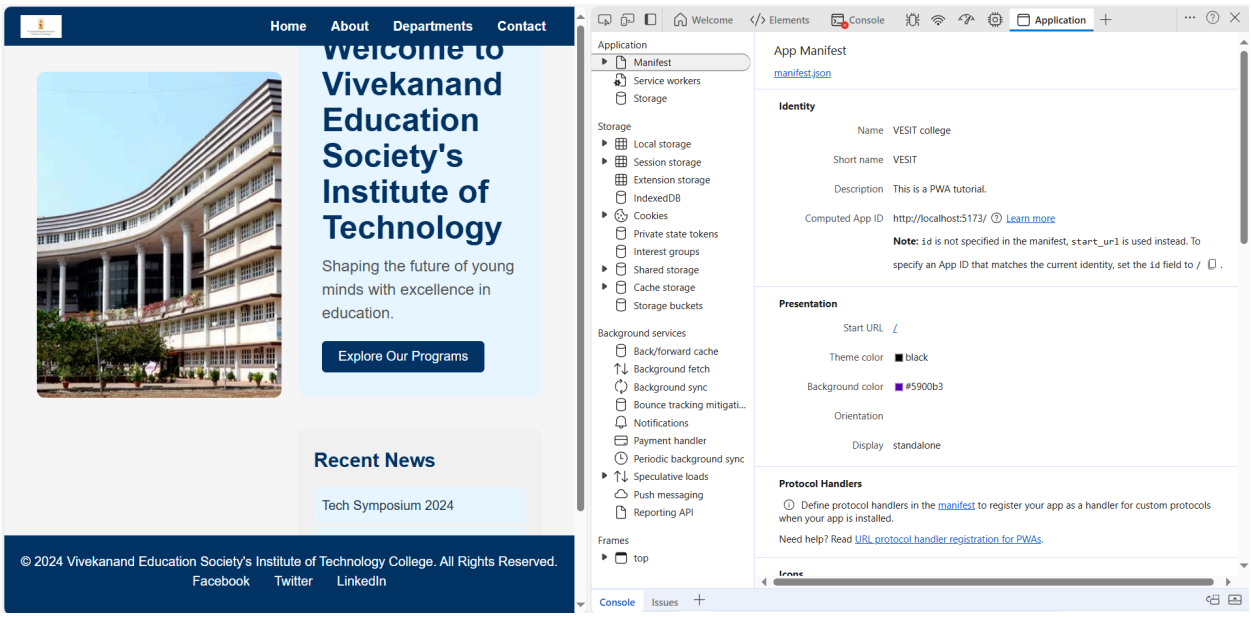
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name=
"apple-mobile-web-app-status-bar"
                        content="#aa7700">
<meta name="theme-color"
                        content="black">

<!-- Manifest File link -->
<link rel="manifest"
                        href="/manifest.json">
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
    <script>
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/serviceworker.js')
    .then(function(registration) {
      console.log('Service Worker registered with scope:', registration.scope);
    })
    .catch(function(error) {
      console.log('Service Worker registration failed:', error);
    });
}
</script>
  </body></html>
```

Project Title: VESIT Website **Roll No.: 14**

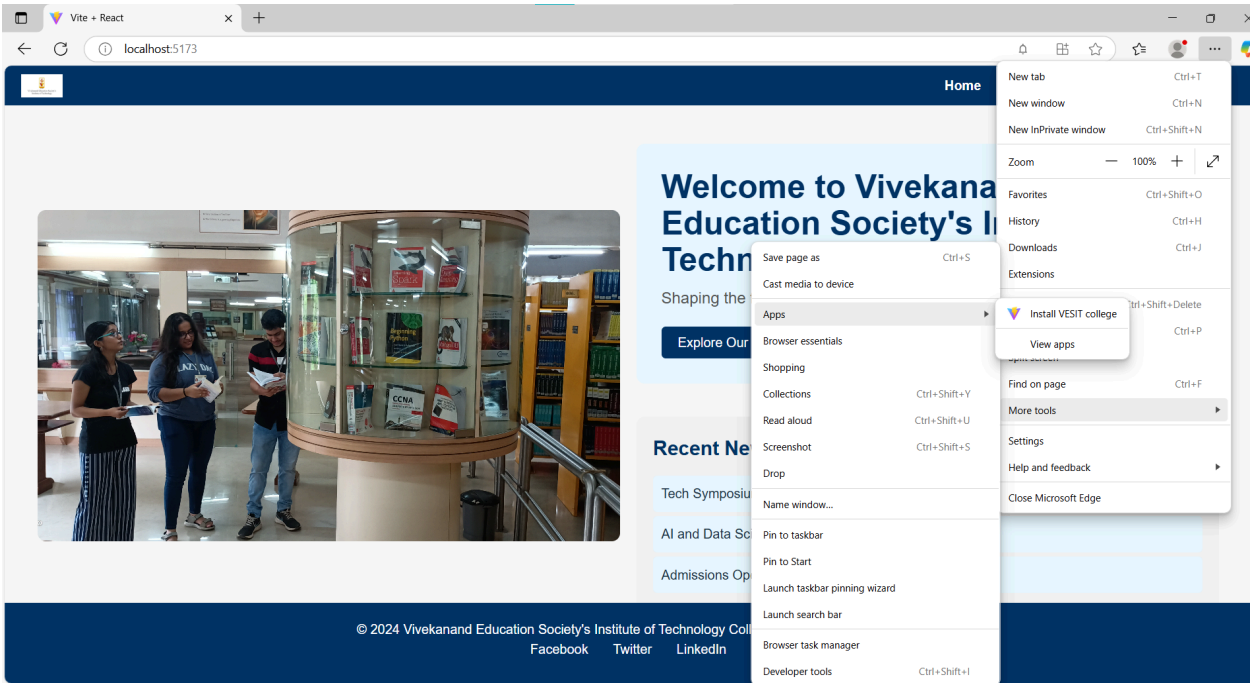
Output :

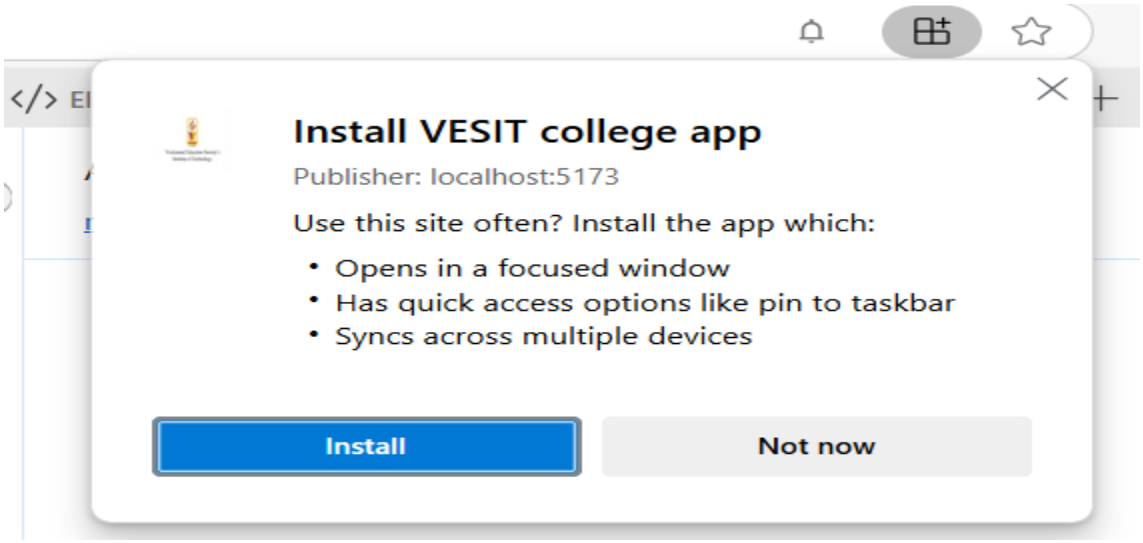
For Edge

open Developer tools options -> Application

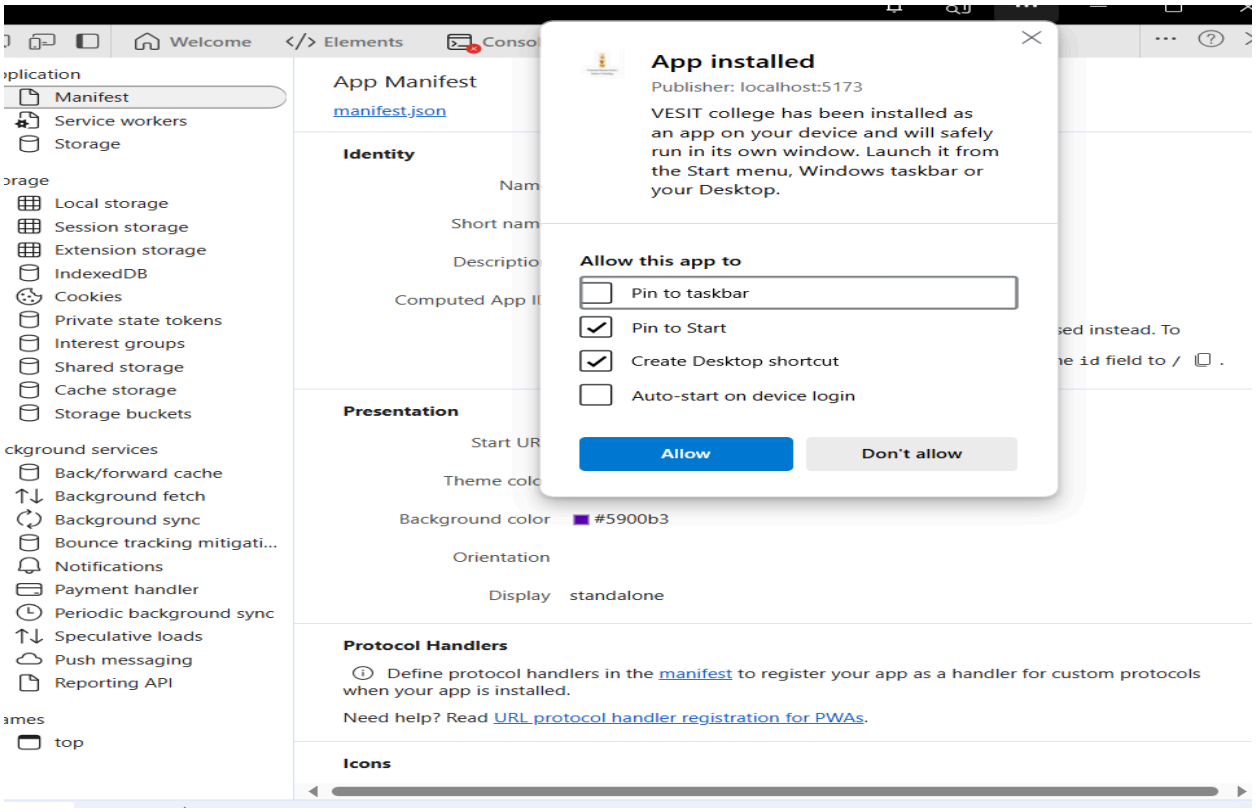


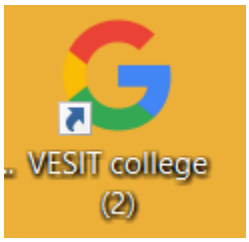
click on 3 dots on top right corner of browser from app option install this site as an app
3 dots -> more tools - > apps -> Install





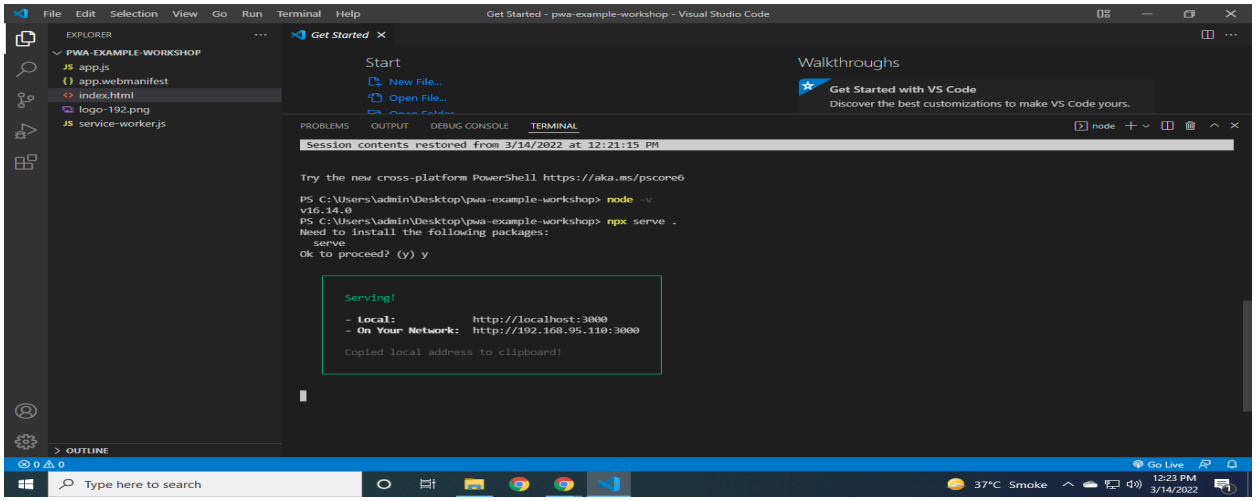
Check create desktop shortcut - > allow

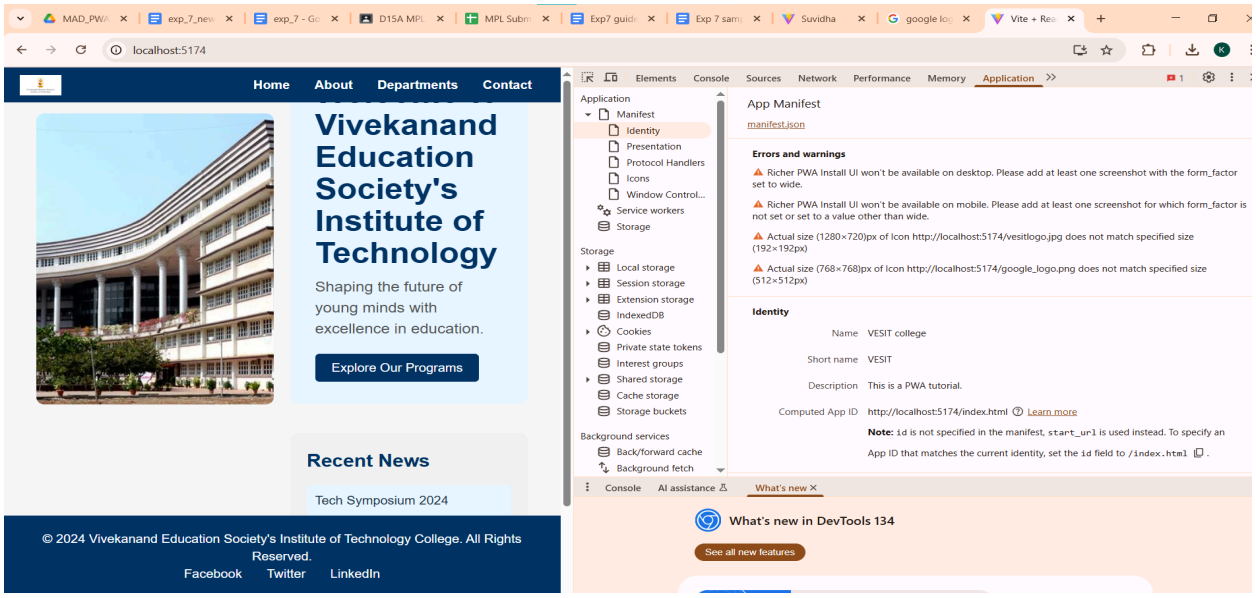


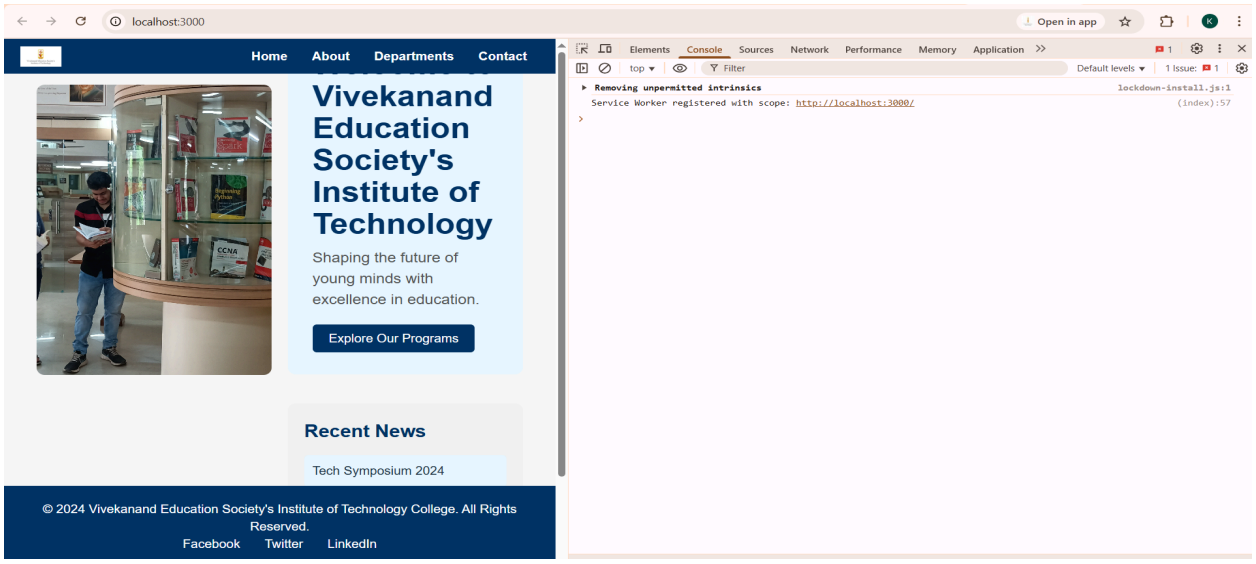


Project Title: VESIT Website **Roll No.: 14**

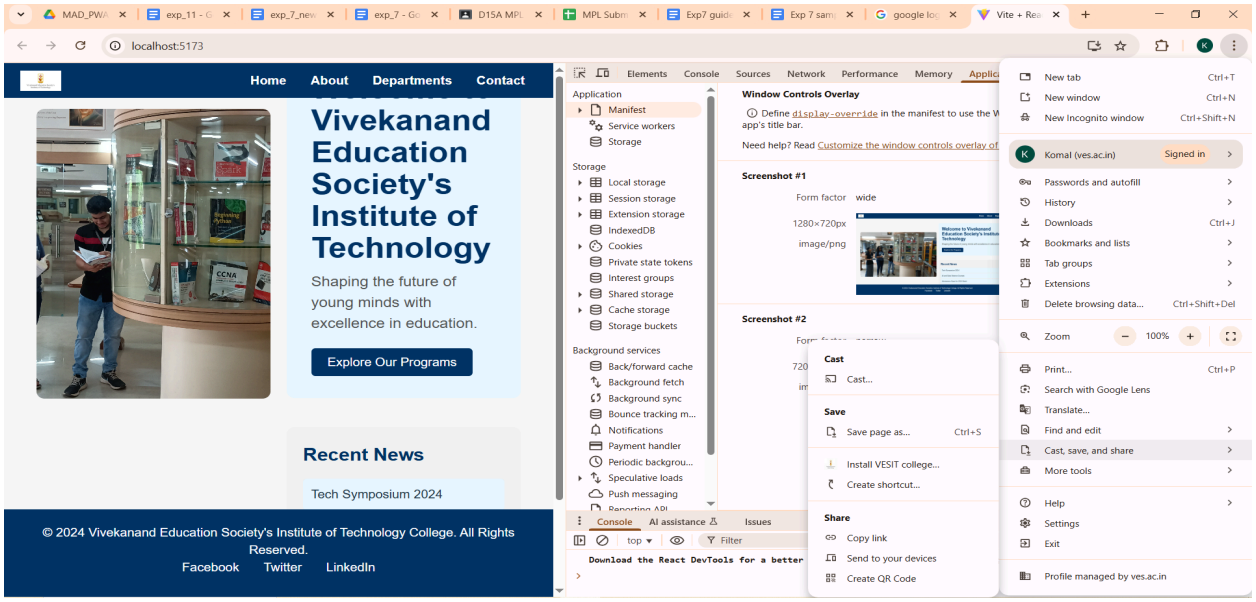
For Chrome

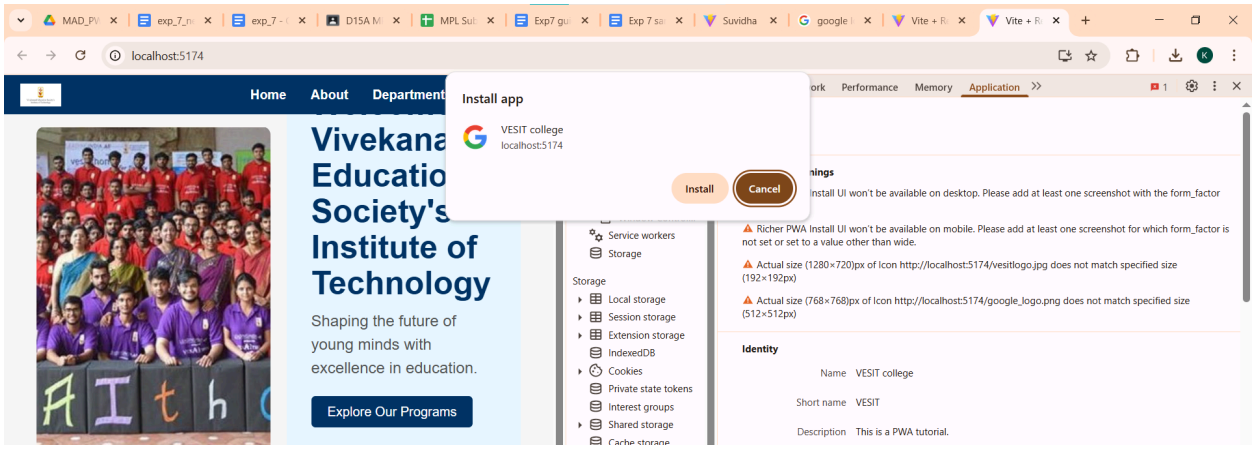


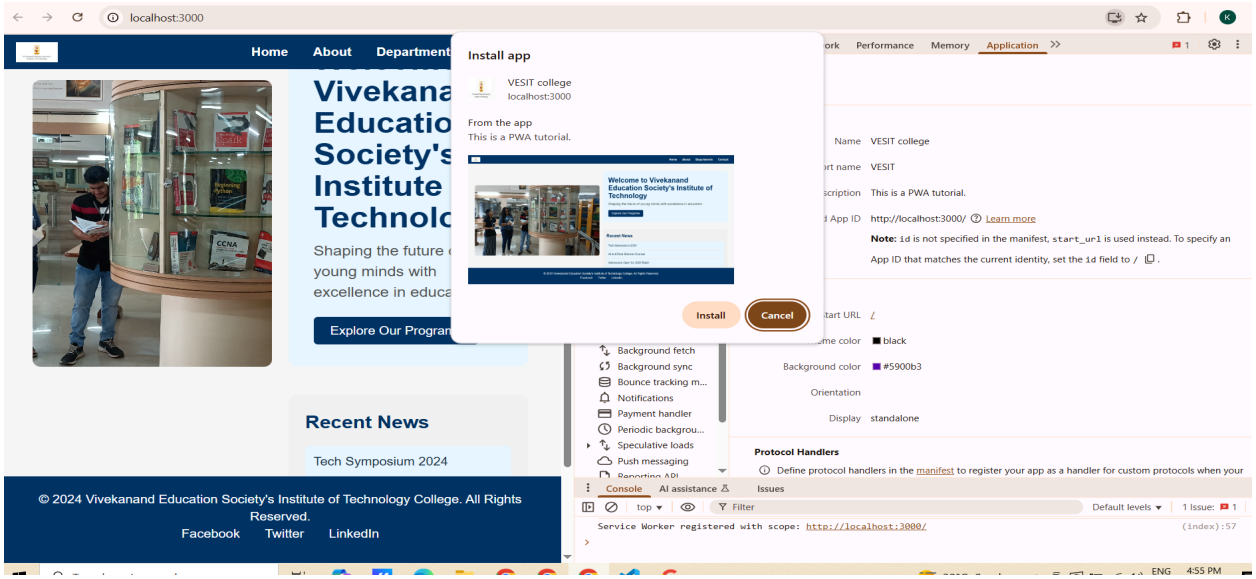




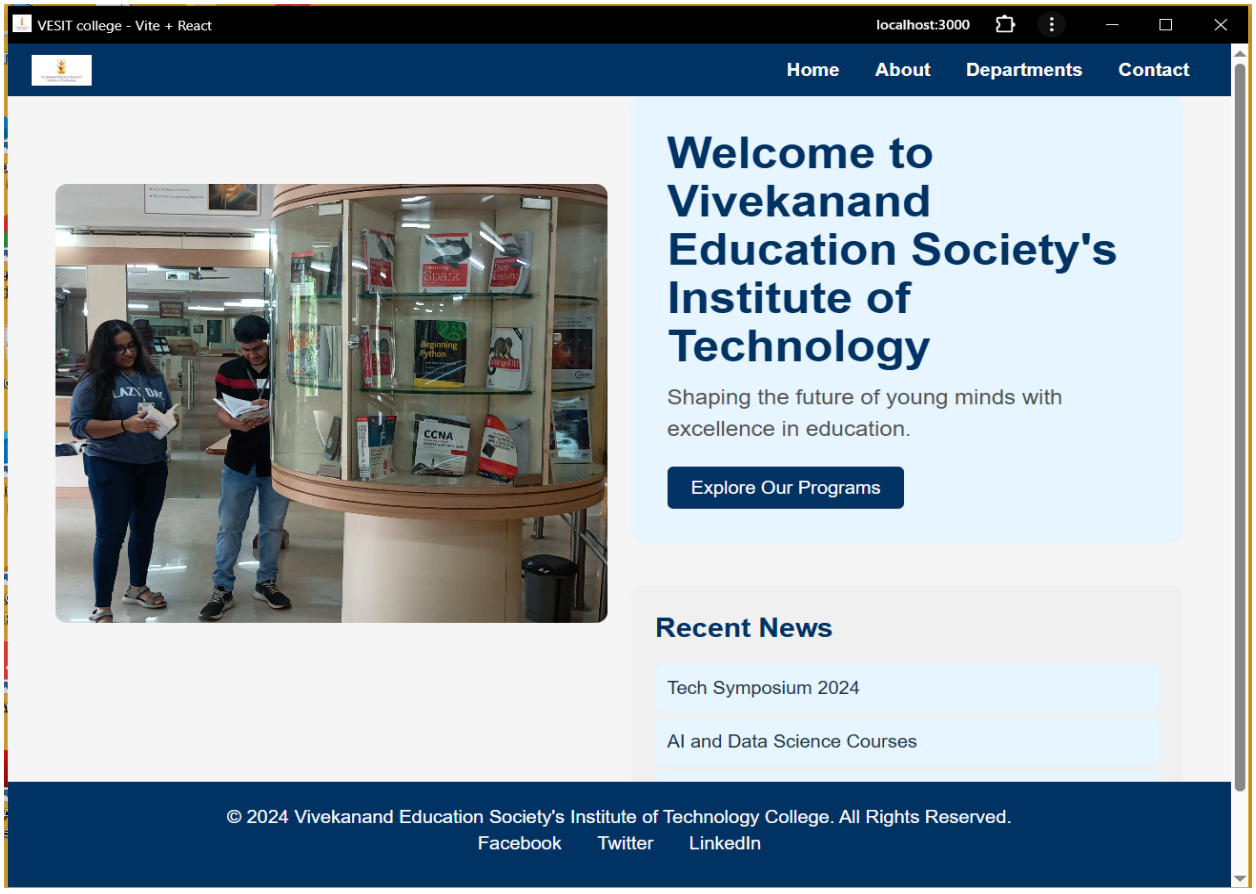
Project Title: VESIT Website **Roll No.: 14**

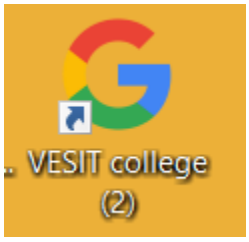






Project Title: VESIT Website **Roll No.: 14**





Conclusion :

Hence, we learnt how to write a metadata of our website PWA in a Web App Manifest File to enable add to homescreen feature.

https://medium.com/@svinkle/start-a-local-live-reload-web-server-with-one-command-72f99bc6e855