



BHARATI VIDYAPEETH



COLLEGE OF ENGINEERING

DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR: 2021-2022

COURSE NAME: MAD & PWA Lab

COURSE CODE	ITL604						
EXPERIMENT NO	08						
EXPERIMENT TITLE.	To write meta data of your application PWA in a Web App manifest file to enable" Add to Home screen features"						
NAME OF STUDENT	Prathamesh Sable						
ROLL NO.	60						
CLASS	TE-IT						
SEMESTER	VI						
GIVEN DATE	16/3/2022						
SUBMISSION DATE	30/3/2022						
CORRECTION DATE							
REMARK							
TIMELY SUBMISSION		PRESENTATION		UNDERSTANDING		TOTAL MARKS	
	04		04		07		15
NAME & SIGN. OF FACULTY		Prof.S.M.Patil					

EXPERIMENT NO:-8

Aim: To write a meta data of application PWA in web - App manifest file to enable “add” to Home Screen Features”

Theory:

Progressive Web Apps (PWAs) are gaining more and more popularity in these days. Maybe you’ve already used a PWA without knowing because many big websites have already enhanced their platforms with PWA features.

A Progressive Web App is a web application which makes use of latest web technologies to make a web application act and feel like an app. This is achieved by making use of web app manifest files and service workers. Using that technologies a PWA is able to close the gap between a classic web application and a desktop. or native mobile application.

Building Blocks of a PWA

To enhance a classic web application with PWA features there are two essential building blocks you should add to your application: a web app manifest file and a service worker. Let’s clarify what’s the prupose of these two elements.

Web App Manifest

The main purpose of the web app manifest file is to provide information about the web application. The JSON structure of that file can contain information such as name, author, icons and description. This information is used to install the web application to the homescreen of a device so that the user can access the application more easily and has an overall app-like experience.

Service Workers

A service worker is a JavaScript file which is added to the project and registered in the browser. The service worker script is then able to run in the background and perform tasks like:

- Recognizing the network state and display a special page if no network / internet connection is available.
- Add application data to the browser cache when online and provide that data when offline.
- Display push notifications to the user when your website is not open

As you can see from that list of use cases, services workers are very powerful and can help to improve the user experience and increase conversion rates significantly.

For now, only Chrome, Firefox and Opera have adequate support for service workers. Of course this means that two big players are missing: Safari and Edge. However, Apple has begun working on adding Service Workers to WebKit, the open source layout engine used by the Safari browser recently. So we can expect to see service worker support in Safari.

Add to Home screen (A2HS for short) is fundamental to the Progressive Web Apps experience as it allows for a full native-app experience, which includes the launching of apps from the user's home screen.

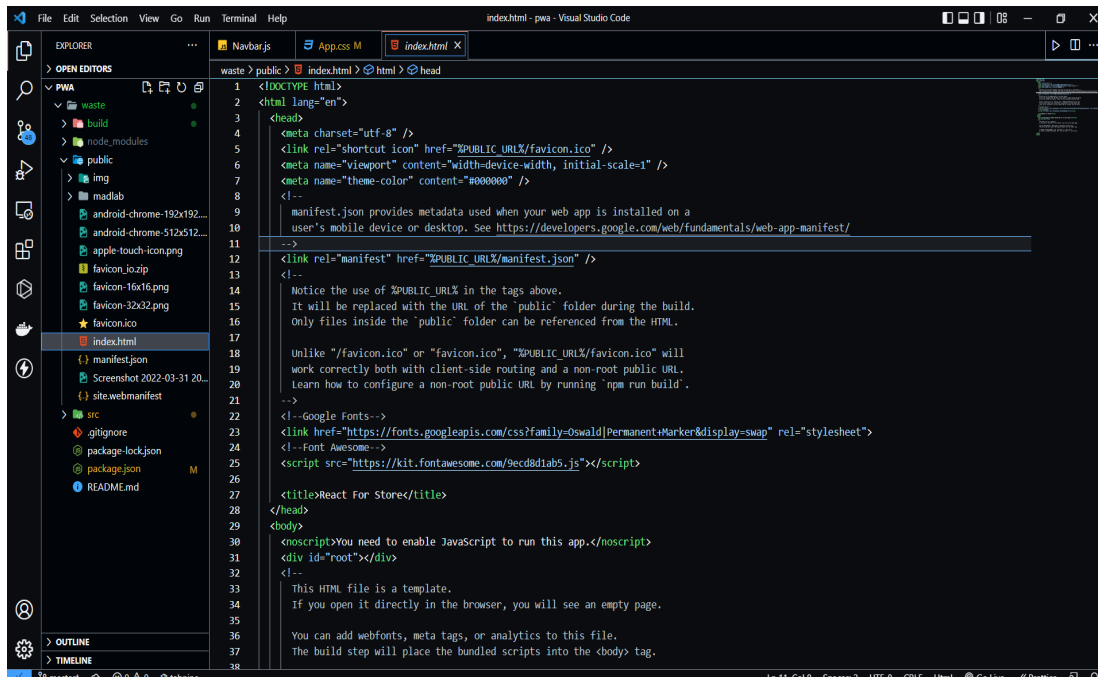
How to make your web app A2HS-enabled on mobile

Installable criteria

For your web app to be installable, it must:

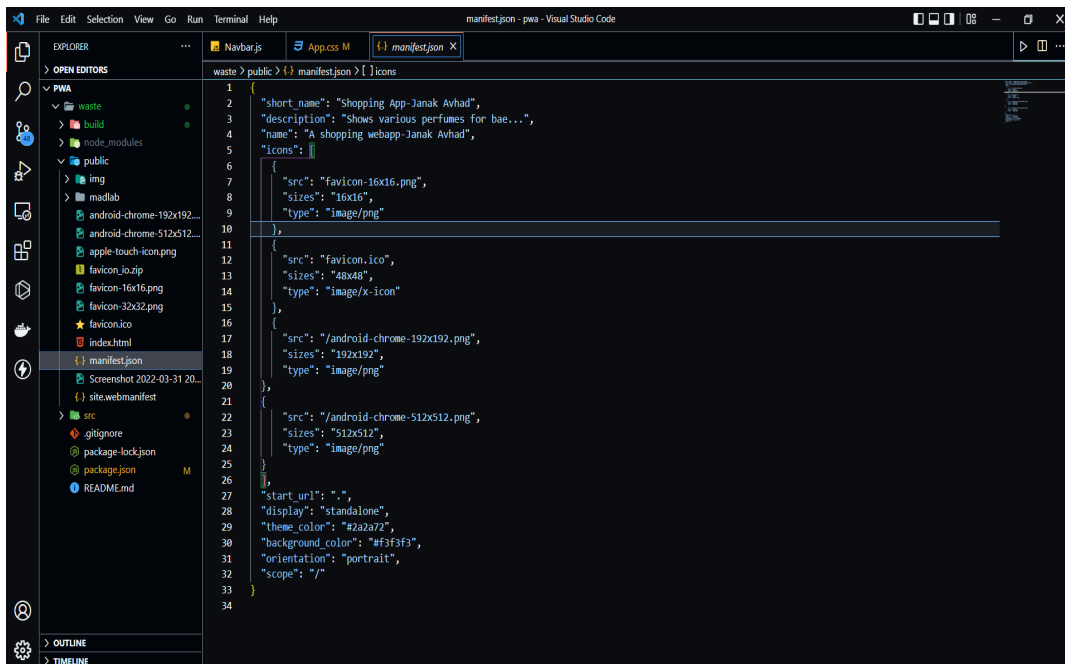
- not already be installed
- meets a user engagement heuristic (the user must interact with the page for at least 30s)
- be served from a secure (HTTPS) connection
- have a service worker registered
- have a properly configured manifest file
- And most importantly, the browser in use must support the `beforeinstallprompt` event. For a full list of browsers that support the `beforeinstallprompt` event, do refer to [CanIUse](#).

OUTPUT



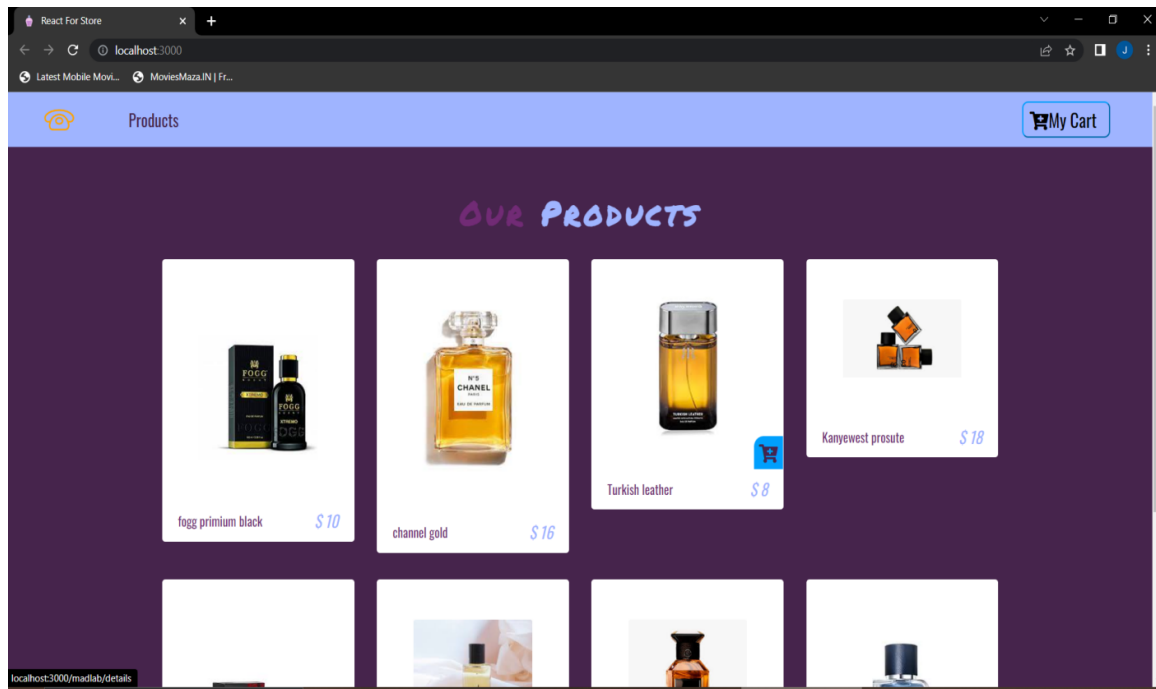
The screenshot shows the Visual Studio Code interface with the 'index.html' file open. The Explorer sidebar on the left shows the project structure, including a 'public' folder with various assets like 'favicon-16x16.png' and 'manifest.json'. The main editor displays the HTML code for 'index.html', which includes a DOCTYPE declaration, HTML language tag, charset, viewport, and theme-color meta tags. It also features a link to 'manifest.json' and a comment explaining the use of 'PUBLIC_URL' placeholders. The body of the page contains a title 'React For Storek', a noscript warning, and a comment about the build process.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico" />
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <meta name="theme-color" content="#000000" />
8     <!--
9       manifest.json provides metadata used when your web app is installed on a
10      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
11    -->
12     <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
13     <!--
14       Notice the use of %PUBLIC_URL% in the tags above.
15       It will be replaced with the URL of the 'public' folder during the build.
16       Only files inside the 'public' folder can be referenced from the HTML.
17
18       Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
19       work correctly both with client-side routing and a non-root public URL.
20       Learn how to configure a non-root public URL by running 'npm run build'.
21     -->
22     <!-- Google Fonts -->
23     <link href="https://fonts.googleapis.com/css?family=Oswald|PermanentMarker&display=swap" rel="stylesheet">
24     <!-- Font Awesome -->
25     <script src="https://kit.fontawesome.com/9ecd8d1ab5.js"></script>
26
27     <title>React For Storek</title>
28   </head>
29   <body>
30     <noscript>You need to enable JavaScript to run this app.</noscript>
31     <div id="root"></div>
32     <!--
33       This HTML file is a template.
34       If you open it directly in the browser, you will see an empty page.
35
36       You can add webfonts, meta tags, or analytics to this file.
37       The build step will place the bundled scripts into the <body> tag.
38     -->
```



The screenshot shows the Visual Studio Code interface with the 'manifest.json' file open. The Explorer sidebar on the left shows the project structure, including a 'public' folder with various assets like 'favicon-16x16.png' and 'manifest.json'. The main editor displays the JSON code for 'manifest.json', which defines the application's metadata, including its name, description, short name, and a list of icons with their sources, sizes, and types. It also specifies the start URL, display mode, theme color, background color, orientation, and scope.

```
1 {
2   "short_name": "Shopping App-Janak Avhad",
3   "description": "Shows various perfumes for bae...",
4   "name": "A shopping webapp-Janak Avhad",
5   "icons": [
6     {
7       "src": "Favicon-16x16.png",
8       "sizes": "16x16",
9       "type": "image/png"
10    },
11    {
12       "src": "favicon.ico",
13       "sizes": "48x48",
14       "type": "image/x-icon"
15    },
16    {
17       "src": "/android-chrome-192x192.png",
18       "sizes": "192x192",
19       "type": "image/png"
20    },
21    {
22       "src": "/android-chrome-512x512.png",
23       "sizes": "512x512",
24       "type": "image/png"
25    }
26  ],
27   "start_url": ".",
28   "display": "standalone",
29   "theme_color": "#2a2a72",
30   "background_color": "#f3f3f3",
31   "orientation": "portrait",
32   "scope": "/"
33 }
34
```



Conclusion:

Thus, we have successfully wrote meta data of your application PWA in a Web App manifest file to enable" Add to Home screen features".