

Experiment No. 7

Aim:- To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.

Theory:-

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.).

Code:-

manifest.json:-

```
{
  "name": "footcap",
  "short_name": "FC",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#5900b3",
  "theme_color": "black",
  "scope": ".",
  "description": "This is a footwear e-commerce website.",
  "icons": [
    {
      "src": "assets/images/img4.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ],
}
```

```
{
  "src": "assets/images/img3.png",
  "sizes": "512x512",
  "type": "image/png"
}
]
```

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

```
<html>
```

```
  <head>
```

```
    <link rel="manifest" href="manifest.json">
```

```
    <title>Footcap - Find your footwear</title>
```

```
    <style>
```

Steps:

Install nodejs

<https://nodejs.org/en/download/>

In cmd

npm -v

npm install -g browser-sync

```
npm install -g live-server
```

Open vs code

Select proper folder where all files are available

Install extensions node js

In vs code terminal type following commands

node -v

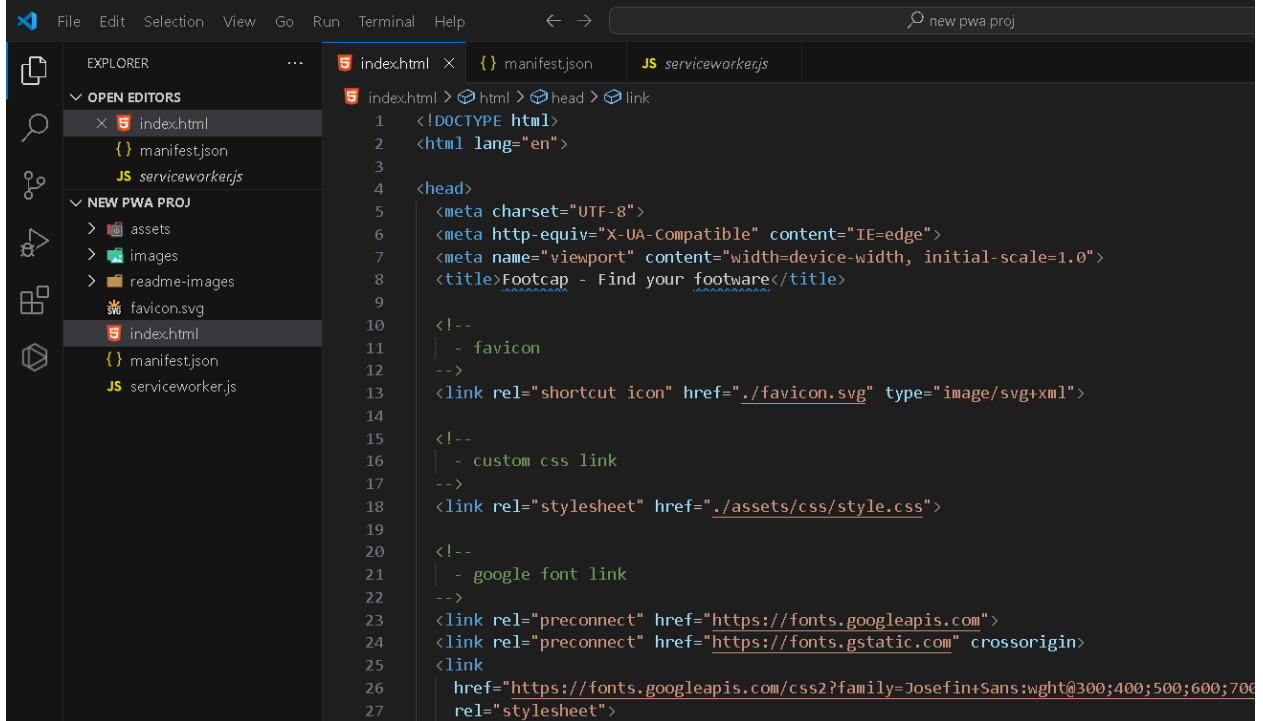
Npx serve .

Reference <https://developer.mozilla.org/en-US/docs/Web/Manifest>

https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

Screenshots:

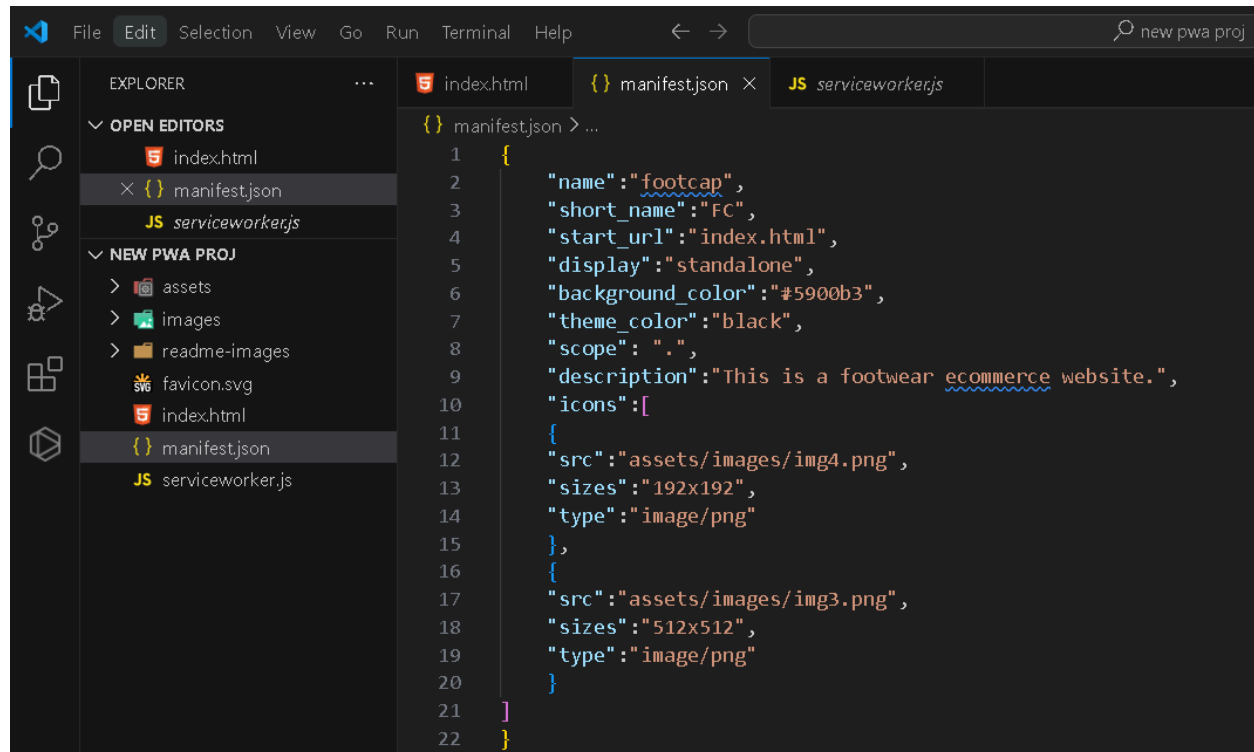
Index.html



The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays the project structure for 'new pwa proj', including 'assets', 'images', 'readme-images', 'favicon.svg', 'index.html', 'manifest.json', and 'serviceworker.js'. The main editor area shows the content of 'index.html', which is an HTML document with a head section containing meta tags for charset, http-equiv, and viewport, a title 'Footcap - Find your footwear', and several link tags for favicon, custom CSS, Google Font, and preconnect. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Footcap - Find your footwear</title>
9
10  <!--
11   | - favicon
12  -->
13  <link rel="shortcut icon" href="./favicon.svg" type="image/svg+xml">
14
15  <!--
16   | - custom css link
17  -->
18  <link rel="stylesheet" href="./assets/css/style.css">
19
20  <!--
21   | - google font link
22  -->
23  <link rel="preconnect" href="https://fonts.googleapis.com">
24  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
25  <link
26    href="https://fonts.googleapis.com/css2?family=Josefin+Sans:wght@300;400;500;600;700
27    rel="stylesheet">
```

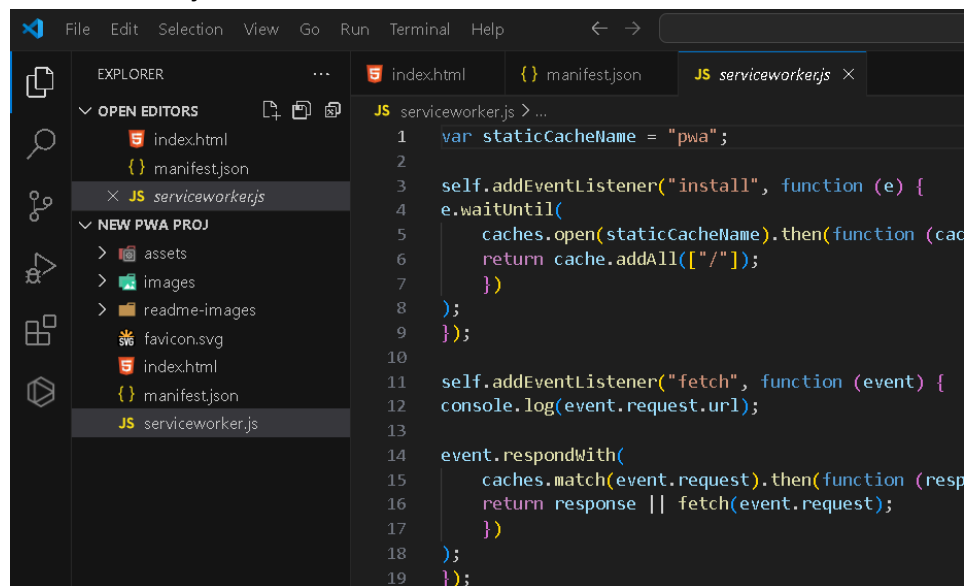
Manifest.json



The screenshot shows the Visual Studio Code interface with a PWA project. The Explorer sidebar on the left shows the project structure under 'NEW PWA PROJ', including folders 'assets' and 'images', and files 'readme-images', 'favicon.svg', 'index.html', 'manifest.json', and 'serviceworker.js'. The 'manifest.json' file is selected and its content is displayed in the main editor. The file defines a web application named 'footcap' with a short name 'FC', a start URL of 'index.html', and a standalone display. It also specifies a background color, theme color, and two icons: 'img4.png' (192x192) and 'img3.png' (512x512). The description is 'This is a footwear ecommerce website.'.

```
{
  "name": "footcap",
  "short_name": "FC",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#5900b3",
  "theme_color": "black",
  "scope": ".",
  "description": "This is a footwear ecommerce website.",
  "icons": [
    {
      "src": "assets/images/img4.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "assets/images/img3.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

serviceworker.js



The screenshot shows the Visual Studio Code interface with the 'serviceworker.js' file selected in the Explorer sidebar. The file content is displayed in the main editor. It defines a static cache named 'pwa' and registers two event listeners: 'install' and 'fetch'. The 'install' listener opens the cache and adds all resources. The 'fetch' listener logs the request URL and responds with the cached resource if available, or fetches it from the network otherwise.

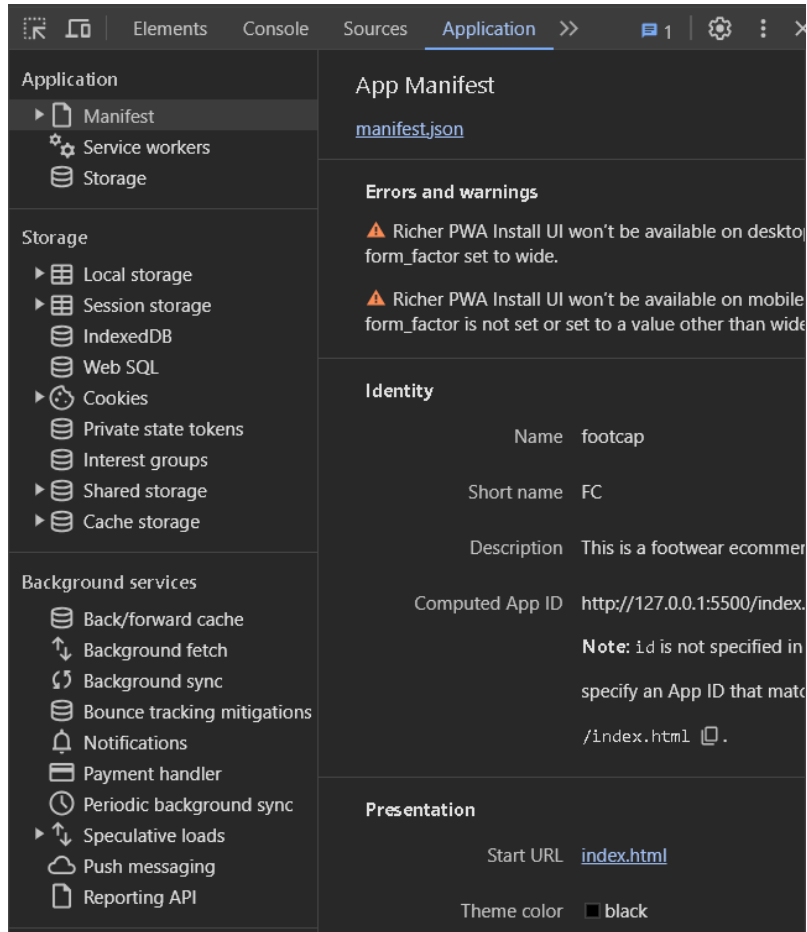
```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
  e.waitUntil(
    caches.open(staticCacheName).then(function (cache) {
      return cache.addAll(["/"]);
    })
  );
});

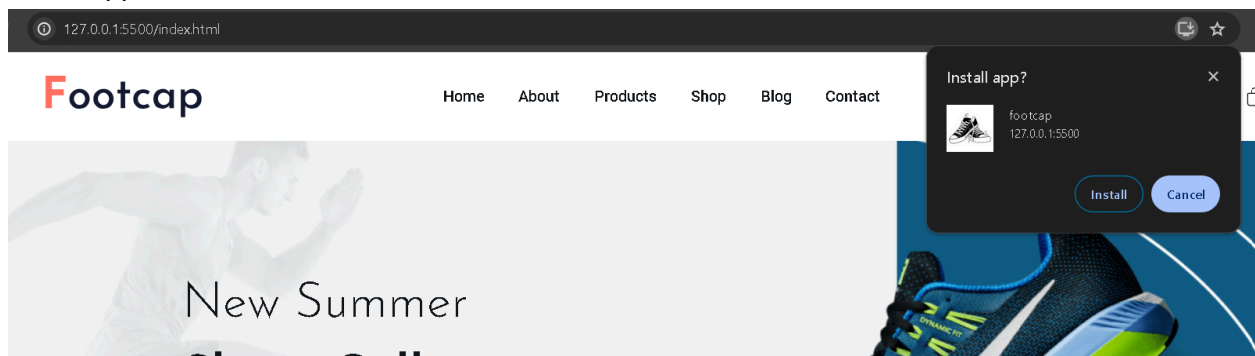
self.addEventListener("fetch", function (event) {
  console.log(event.request.url);

  event.respondWith(
    caches.match(event.request).then(function (response) {
      return response || fetch(event.request);
    })
  );
});
```

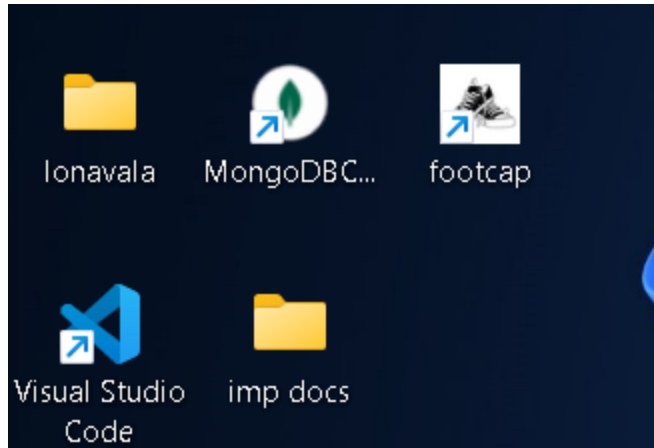
App manifest



Install app



App Icon on desktop

**Conclusion:**

In this experiment, the aim was to enhance the user experience of an ecommerce Progressive Web App (PWA) by enabling the "add to homescreen" feature through the use of metadata in a web app manifest file. By defining key metadata such as the app's name, icons, and start URL in the manifest file, we provided users with a seamless way to add the ecommerce PWA to their device's homescreen, similar to a native app installation. This approach not only improves accessibility to the ecommerce platform but also enhances user engagement by making the app readily available with a single tap, fostering a more immersive and convenient shopping experience.