

## **Experiment No. 7**

**Name : Kunal Pal**

**Div/Roll : D15B/50**

**Subject : PWA LAB**

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

### **Regular Web App:**

- A regular web app is a traditional website accessed through a browser.
- It relies on a server for most of its functionality and data processing.
- It does not necessarily offer offline access, relying on an active internet connection.
- It may not provide features like push notifications or access to device hardware.
- It does not have a manifest file defining its metadata or the ability to be installed on a device's home screen.

### **Progressive Web App (PWA):**

- A Progressive Web App (PWA) is a type of web application that provides a more app-like experience for users.
- It is built with modern web technologies like Service Workers, Web App Manifests, and HTTPS.
- PWAs offer responsive design, adapting to various screen sizes and orientations.
- They can work offline or with a poor internet connection, using Service Workers to cache content.
- PWAs can send push notifications, enabling engagement even when the app is not open.
- They have a manifest file (manifest.json) that defines metadata such as the app's name, icons, colors, and start URL.

- Users can "install" a PWA on their device's home screen, making it accessible like a native app, with an icon and full-screen mode.
- PWAs are discoverable and can be indexed by search engines, improving visibility.
- They provide a seamless, fast, and reliable user experience, leading to higher user engagement and retention.

### **Differences: Offline Functionality:**

- **Regular Web Apps:**
- Typically rely on an active internet connection to function properly.
- When the user loses connectivity, they may see error messages or a lack of new data.
- The app's functionality is limited when offline, as it cannot access previously loaded content.
- **Progressive Web Apps (PWAs):**
- Use Service Workers to cache content, allowing them to work offline or in low-connectivity situations.
- Users can continue to browse content, view pages, and interact with the app even without an internet connection.
- Updates made while offline are synchronized when the device reconnects to the internet.

### **Installation:**

- **Regular Web Apps:**
- Accessed by typing the URL into a web browser.
- Do not have an option for installation on the device's home screen.
- **Progressive Web Apps (PWAs):** • Can be "installed" on the user's device, creating an icon on the homescreen.
- This installation is done through a browser prompt or a manual option from the browser menu.
- Once installed, PWAs launch in full-screen mode, resembling native mobile apps.

## **Push Notifications:**

- Regular Web Apps:
  - Lack the ability to send push notifications to users.
  - Engagement with users outside of the app is limited.
- Progressive Web Apps (PWAs):
  - Have the capability to send push notifications, even when the app is not actively open.
  - This feature helps in re-engaging users, announcing updates, promotions, or relevant information.

## **Metadata and Manifest:**

- **Regular Web Apps:**
  - Do not have a manifest file (manifest.json) defining metadata.
  - Metadata like icons, colors, and start URLs are typically not specified.
- Progressive Web Apps (PWAs):
  - Use a manifest.json file to define metadata about the app.
  - Developers can specify the app's name, icons for different device sizes, background color, theme color, and more.
  - This metadata allows the PWA to appear more like a native app when installed on the
  - homescreen.

## **Performance and Optimization:**

- Regular Web Apps:
  - Performance is based on the browser and device capabilities.
  - May not always provide a smooth, app-like experience due to varying browser support.
- Progressive Web Apps (PWAs):
  - Optimized for performance and reliability.
  - Utilize modern web technologies such as Service Workers and efficient caching strategies.

- Offer a faster, more responsive, and smoother user experience, especially on mobile devices.
- Can be added to the user's home screen, launching quickly and providing a native-like feel.

### **Discoverability and Indexing:**

- Regular Web Apps:
  - Discoverability by search engines can be limited.
  - May not appear in app stores or be easily found by users searching for similar apps.
- Progressive Web Apps (PWAs):
  - Fully discoverable by search engines, improving visibility.
  - Can be indexed and ranked in search results, similar to regular websites.
  - Users searching for related topics may discover PWAs through search engine results.

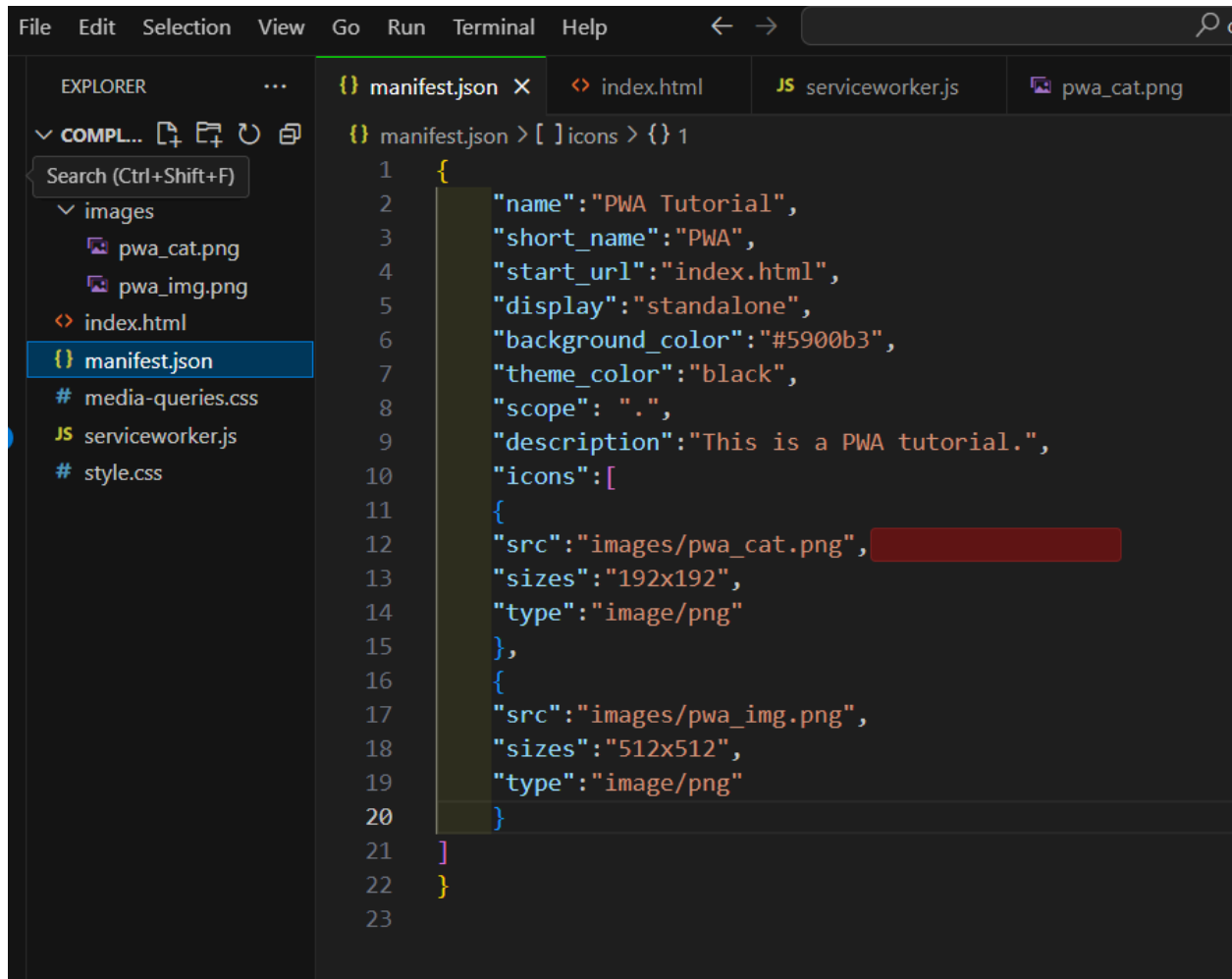
### **Implementation:**

**The below steps have to be followed to create a progressive web application:**

**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 will be created in the next step.

**Step 2:** Create a manifest.json file in the same directory. This file 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 the size of the icons are also defined in this file.

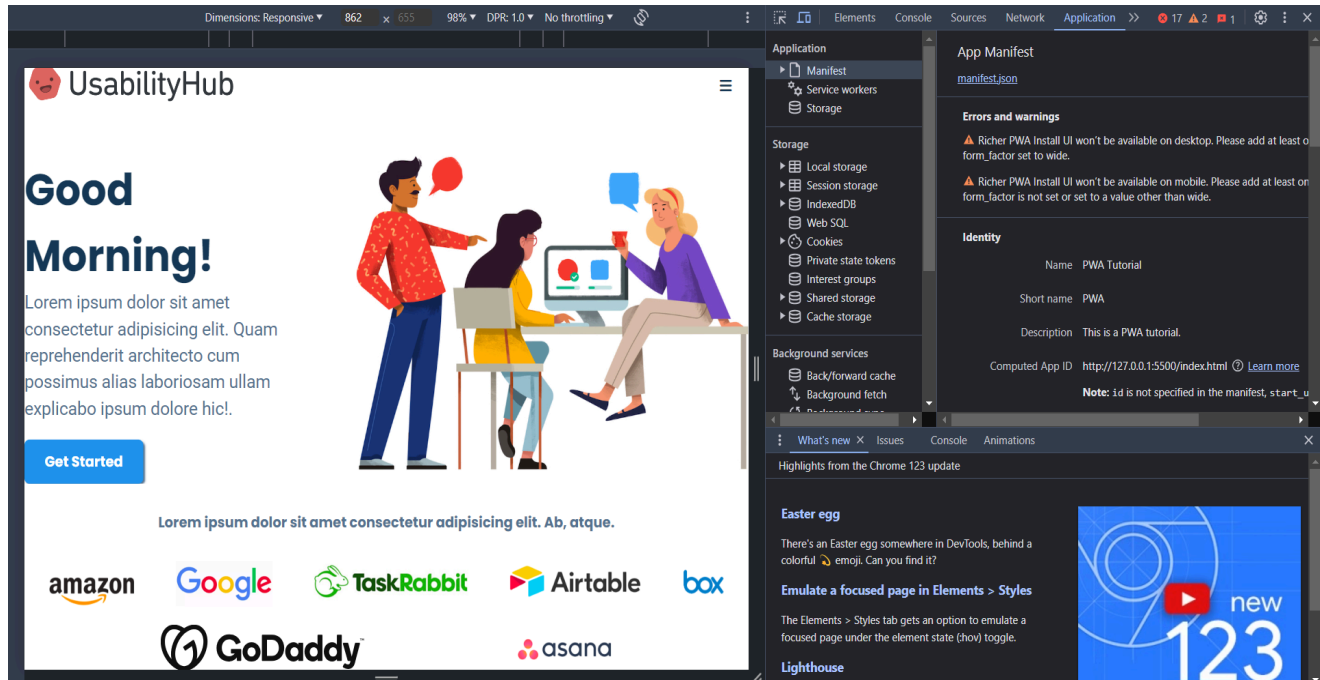
1) This is my manifest.json file with the two images :



The image shows a screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left displays a project structure with a folder named 'images' containing two files: 'pwa\_cat.png' and 'pwa\_img.png'. Below the 'images' folder are files 'index.html', 'manifest.json' (which is selected and highlighted in blue), 'media-queries.css', 'serviceworker.js', and 'style.css'. The main editor area shows the content of 'manifest.json'. The file has tabs for 'manifest.json', 'index.html', 'serviceworker.js', and 'pwa\_cat.png'. The JSON content is as follows:

```
1  {
2      "name": "PWA Tutorial",
3      "short_name": "PWA",
4      "start_url": "index.html",
5      "display": "standalone",
6      "background_color": "#5900b3",
7      "theme_color": "black",
8      "scope": ".",
9      "description": "This is a PWA tutorial.",
10     "icons": [
11         {
12             "src": "images/pwa_cat.png",
13             "sizes": "192x192",
14             "type": "image/png"
15         },
16         {
17             "src": "images/pwa_img.png",
18             "sizes": "512x512",
19             "type": "image/png"
20         }
21     ]
22 }
```

2) Here you can see the output of the images in inspect>application section



3) This is my servicesworks.js file

```
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);
    })
};
});

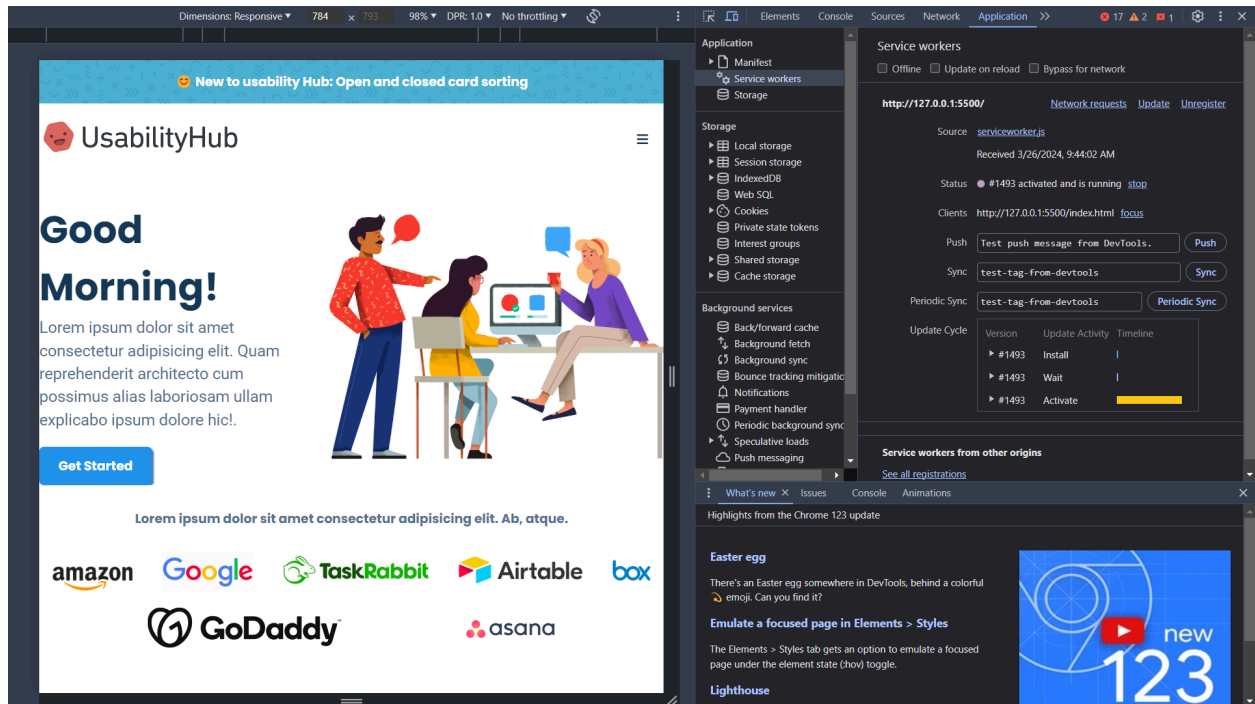
```

4) This is index file of the website

```

<!-- index.html -->
2  <html lang="en">
11 <body>
257 </div>
258
259 <script src="https://kit.fontawesome.com/7039b24af9.js" crossorigin="anonymous"></script>
260
261 <script>
262     window.addEventListener('load', () => {
263         registerSW();
264     });
265
266     // Register the Service Worker
267     async function registerSW() {
268         if ('serviceWorker' in navigator) {
269             try {
270                 await navigator
271                     .serviceWorker
272                     .register('serviceworker.js');
273             }
274             catch (e) {
275                 console.log('SW registration failed');
276             }
277         }
278     }
279 </script>
280
281 <!-- <script src="serviceworker.js"></script> -->
282 </body>
283 </html>

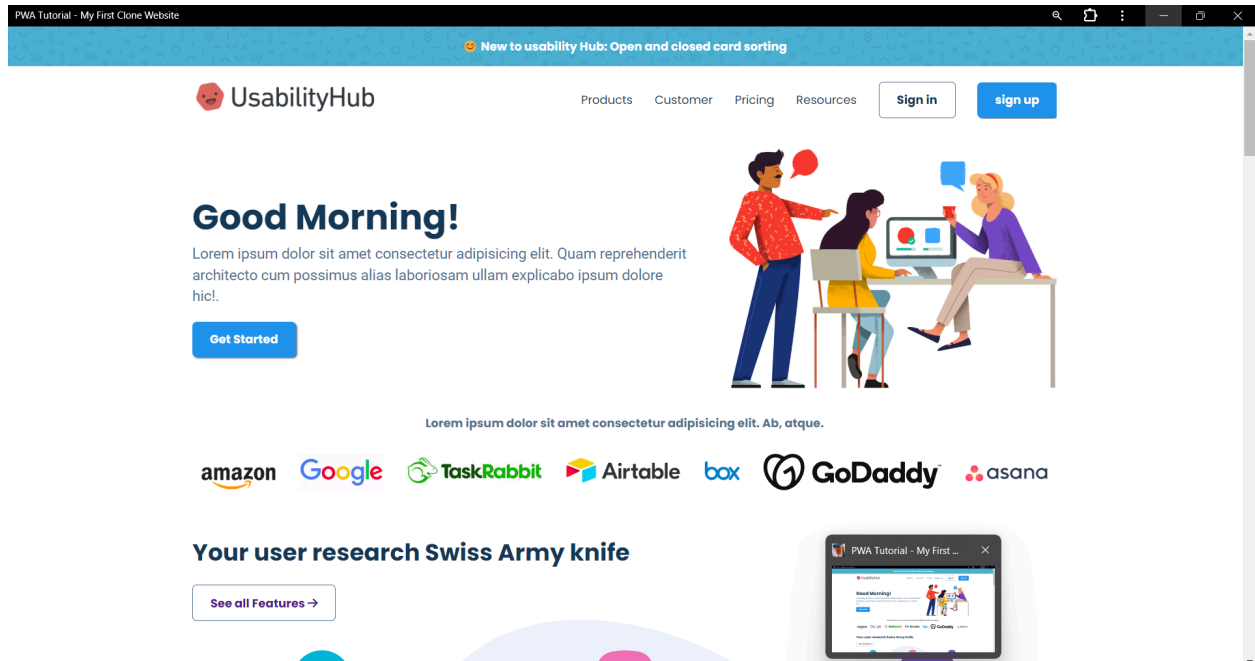
```



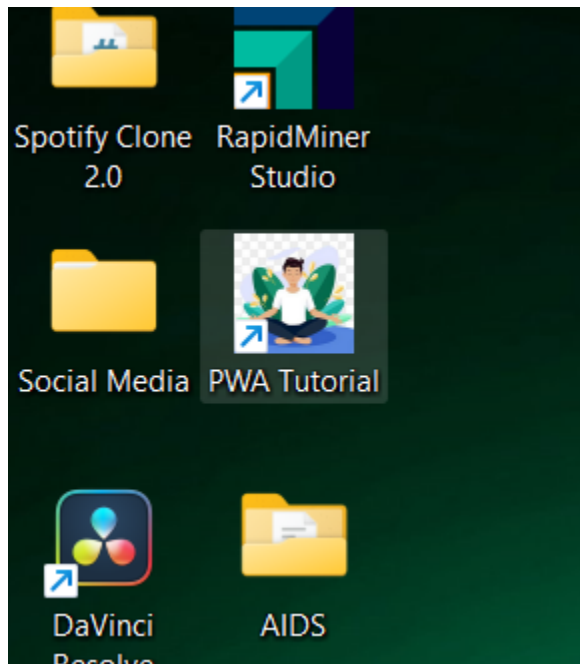
5) Know you can download this application offline in your desktop :







6) Downloaded App in the desktop



**Conclusion :** This experiment involved creating a straightforward HTML E-commerce home page showcasing products with images, titles, prices, descriptions, and "Add to Cart" buttons. This foundational layout offers a starting point for developing a more robust E-commerce platform, with the potential for expansion into categories, navigation.