

## PWA EXP 1

**Aim:** Crafting Metadata for Progressive Web Applications (PWAs) to Enable "Add to Homescreen" Functionality

### **Introduction to Progressive Web Applications (PWAs):**

Progressive Web Applications (PWAs) represent a modern approach to web development, leveraging the capabilities of modern browsers to deliver an app-like experience across various devices. Built with standard web technologies like HTML, CSS, and JavaScript, PWAs offer a range of advantages over traditional web apps.

**Distinguishing PWAs from Regular Web Apps:**

PWAs differ significantly from regular web apps in several key aspects, enhancing user experience and functionality:

#### **Offline Functionality:**

PWAs employ service workers to enable offline functionality, allowing the application to remain functional even with limited or no internet connectivity. Service workers can cache assets and data, ensuring a seamless user experience regardless of network conditions.

#### **App-like User Interface:**

PWAs offer an immersive user interface similar to native mobile apps. They can be installed on the device's homescreen, launched in full-screen mode, and utilize features like push notifications, providing users with a native app feel.

#### **Improved Performance:**

Optimized for performance, PWAs deliver faster loading times and smoother interactions compared to traditional web apps. Techniques such as lazy loading, code splitting, and caching minimize loading times and enhance responsiveness.

#### **Discoverability and Shareability:**

PWAs are discoverable through search engines and shareable via URL links, facilitating easy access and content sharing. Web app manifests and service worker registration scopes further enhance the integrated user experience.

#### **Cross-platform Compatibility:**

Designed to work seamlessly across different platforms and devices, PWAs adapt to various screen sizes and orientations, ensuring a consistent user experience across devices without the need for separate native apps.

### **Steps to Create a Progressive Web Application:**

#### **Create HTML Page:**

Begin by creating an HTML page serving as the starting point of the application. Include a link to the manifest.json file, which contains essential information about the web application.

html

Anket Kadam

D15B/26

Copy code

<!-- HTML code here -->

```
index.html X
index.html > ...
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 |
9 |   <!--
10 |    - primary meta tags
11 |   -->
12 |   <title>Shoppie - Man summer collection</title>
13 |   <meta name="title" content="Shoppie - Man summer collection">
14 |   <meta name="description" content="This is an eCommerce html template made by codewithsadee">
15 |
16 |   <!--
17 |    - favicon
18 |   -->
19 |   <link rel="shortcut icon" href="./favicon.svg" type="image/svg+xml">
20 |
21 |   <!--
22 |    - google font link
23 |   -->
24 |   <link rel="preconnect" href="https://fonts.googleapis.com">
25 |   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
26 |   <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600&display=swap" rel="stylesheet">
27 |   <link rel="stylesheet" href="./assets/fonts/font.css">
28 |
29 |   <!--
30 |    - custom css link
31 |   -->
32 |   <link rel="stylesheet" href="./assets/css/style.css">
33 |
34 |   <!--
35 |    - preload images
36 |   -->
37 |   <link rel="preload" as="image" href="./assets/images/hero-banner.png">
38 |
39 |</head>
```

### Create Manifest File (manifest.json):

Develop a manifest.json file containing metadata about the web application, including the application name, starting URL, theme color, and icons. Define the source and size of icons within the manifest file.

## Anket Kadam

### D15B/26

```
manifest.json > ...
1  {
2    "name": "Shopiee",
3    "short_name": "Shopiee",
4    "start_url": "index.html",
5    "display": "standalone",
6    "background_color": "#FFFFFF",
7    "theme_color": "#FF5722",
8    "description": "Your ultimate destination for trendy e-commerce shopping!",
9    "icons": [
10     {
11       "src": "img/icon-192x192.png",
12       "sizes": "192x192",
13       "type": "image/png"
14     },
15     {
16       "src": "img/icon-512x512.png",
17       "sizes": "512x512",
18       "type": "image/png"
19     }
20   ]
21 }
```

### Link Service Worker to HTML:

Add a JavaScript script within the HTML file to link the service worker file, enabling its functionality within the application.

```
<script>
window.addEventListener('load', () => {
  registerSW();
});

async function registerSW() {
  if ('serviceWorker' in navigator) {
    try {
      await navigator.serviceWorker.register('serviceworker.js');
      console.log('Service Worker registered successfully');
    } catch (error) {
      console.log('Service Worker registration failed:', error);
    }
  }
}
</script>
```

Storage

Local storage

Session storage

IndexedDB

Web SQL

Cookies

Private state tokens

Interest groups

Shared storage

Cache storage

Background services

Back/forward cache

Background fetch

Background sync

Source serviceworker.js 2

Received 19/03/2024, 23:10:59

Status ● #15 is redundant

Push  Push

Sync  Sync

Periodic Sync  Periodic Sync

Update Cycle

Version	Update Activity	Timeline
▶ #15	Install	
▶ #15	Activate	

SHOPPIE

assets

css

manifest.json > ...

1 {

2 "name": "Shopiee",

### **Conclusion:**

Crafting metadata for PWAs is essential for enabling features like "Add to Homescreen" functionality, enhancing accessibility, and improving user experience. By adhering to best practices and accurately defining metadata properties within the manifest file, developers can create PWAs that offer seamless integration, cross-device compatibility, and enhanced functionality.