Progressive Web Applications / Converting an App into A Progressive Web App

Name

Institutional affiliation

Course

Instructor

Date

Converting an App into A Progressive Web App

Progressive Web Applications (PWAs) refer to web applications that are progressive in nature. The term progressive refers to the continued alterations to the system intended to better the customer experience from the initial installation through to their current position (MDN Contributors, 2021). Usually, such alterations are guided by the design of a web browser, the functionalities and the characteristics of the application (Behl & Raj, 2018). Most browsers are increasingly being customized to incline towards converting normal web apps to PWA on the merits of their responsiveness to accommodate features such as vibration-on-click action, discoverability, the ability to be installed on more mediums, offline modes, and safety by utilizing HTTPS connections through a Secure Sockets Layer (SSL) (MDN Contributors, 2021). The above characteristics uniquely position PWAs in the modern internet landscape by narrowing the gap between real native applications and web apps. This clarifies the emphasis on the conversion of such apps into progressive apps.

Converting A Normal Web App to A Progressive Web App

Any website can be converted into being Progressive by adhering to and adopting certain concepts and technologies. The article by Biørn-Hansen et al., (2017) narrows down the steps as follows

i. Ensuring the app is responsive and loads most of the resources internally. The app should avoid fetching resources from other external sources.

ii. Introduces a manifest and a service worker.

Step 1

Below is a sample manifest file and service worker.

```
{
"lang" : "en",
"name" : "AppName",
"display" : "fullscreen",
"start_url" : "Link to launch page",
"short_name" : "AppShortName",
"description" : "App Description",
"orientation" : "portrait",
"background_color": "#000000",
"theme_color": "#ffffff",
"generated" : "true",
  "icons": [
    {
      "src": "app/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    }
  ]
}
```


Manifest.json file

This file explains the details of the app, defining the colors and app icons. To identify a site as a PWA, the JSON file is included in the Html page in between the head element using the Meta tag “<link rel="manifest" href="./manifest.json">” allowing the browser to identify it as an

application. However, the installation of the app would not proceed, as the service worker is still missing.

Step 2. Serviceworker.js

Service workers are scripts that allow apps to function offline by intercepting network requests to deliver programmatic or cached responses. Service workers can receive push notifications and synchronize data in the background even when the app is not running, and together with Web App Manifests, allow users to install PWAs to their devices' home screens. To build up a Service worker, the following codes jointly work together (Steiner, 2018). The Service worker script must then be included in all pages of the application.

To begin, two variables must be set. The variables will help define the static and dynamic cache versions to be used, as explained later. In this case, the precache-v-1.0 and runtimeCache-v-1.0 versions respectively are used.

```
const staticCacheName = 'precache-v1.0';
const dynamicCacheName = 'runtimeCache-v1.0';
```

Code to Register the Service Worker

```
if ('serviceWorker' in navigator) {
  try {
    await navigator.serviceWorker.register('./serviceworker.js');
  } catch (e) {
    console.error('Unable to register service worker.', err);
  }
}
```

Code to Install, Cache, And Serve Static Resources to Allow for Offline Mode

The precacheAssets variable is first and contains all resources to be cached. For example, the variable precacheAssets below is set to cache three CSS files.

```
const precacheAssets = [
    'fonts/main.css',
    'css/style.css',
    'css/reset.css'
];
self.addEventListener('install', async event => {
  const cache = await caches.open(staticCacheName);
  await cache.addAll(precacheAssets);
});
```

Code to Fetch and Serve Cached Assets

```
// Fetch Event
                self.addEventListener('fetch', function (event) {
                        event.respondWith(
                                caches.match(event.request).then(cacheRes => {
                                        return cacheRes || fetch(event.request).then(response => {
                                                return
caches.open(dynamicCacheName).then(function (cache) {
                                                        cache.put(event.request, response.clone());
                                                        return response;
                                                })
                                        });
```

```
        }).catch(function() {
            // Fallback Page if there is no Internet Connection
            return caches.match('fallback.html');
        })
    );
});
```

Accommodating these elements converts the app as an installable which works as a Progressive Web App. To test it, the Chrome developer tools or live server and a domain with HTTPS support can be used. Either of these will prompt the installation of the app.

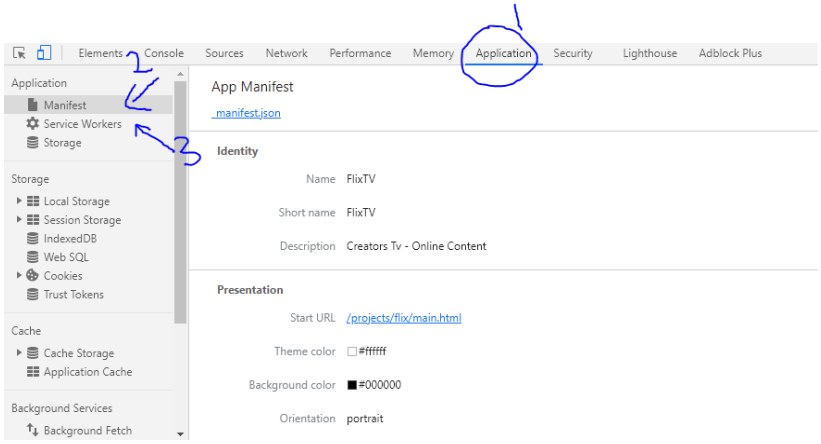
Demonstration of The Installation

Using a sample real live app and the Chrome developer options, the installation would proceed as follows. Below are the steps:

i. Inspect the website by right-clicking then selecting *Inspect*.

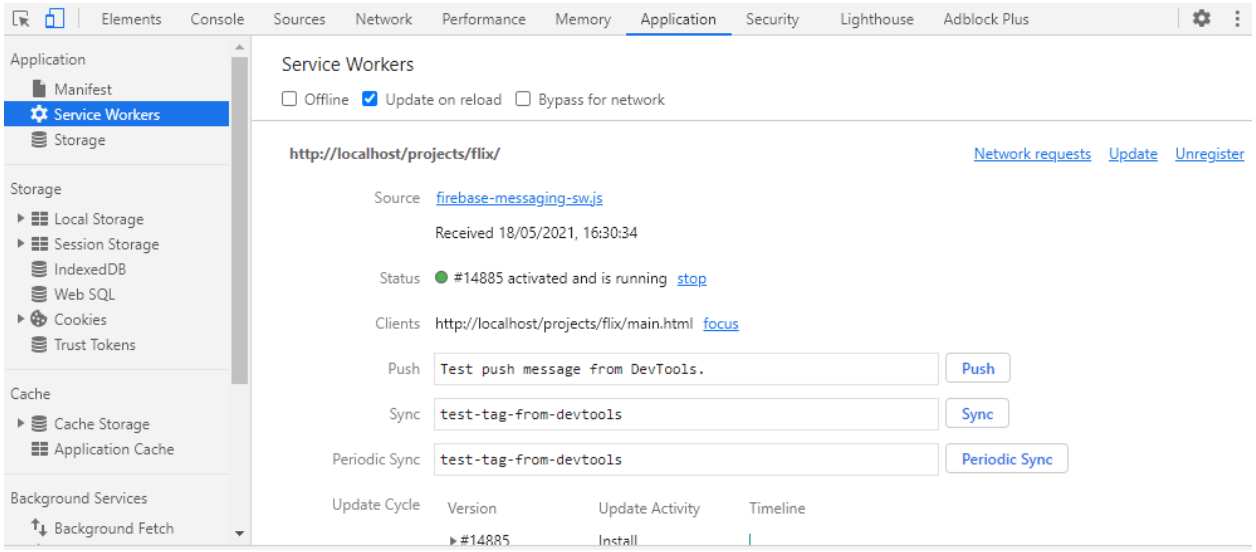
ii. Head to the application tab in the top right corner as indicated in figure (1). The browser helps to determine if the manifest file is read and the site is identified as an application, as Figure (2) shows.

Figure 1. Google Developer Options – Manifest



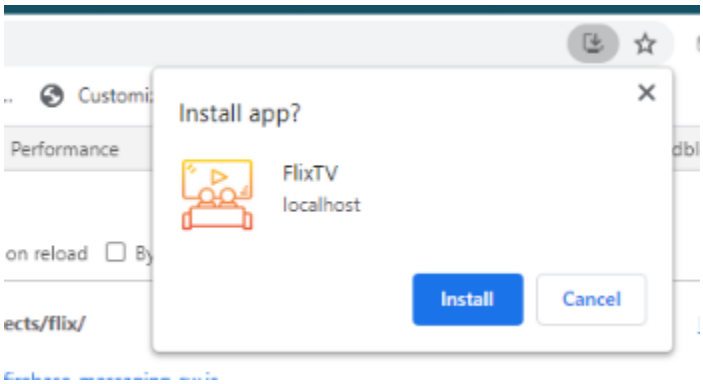
By clicking on *Service Workers* on the left panel, as shown on figure 2, it is possible to detect if a service worker exists.

Figure 2. Google Developer Options – Service Worker



To verify that the application is installable, an install icon will appear on Chrome, as figure 3 demonstrates.

Figure 3. Google Developer Options - Installable Prompt



Conclusion

From a commercial standpoint, PWAs are preferred since they cut down the costs of maintaining and developing IOS, Android and traditional web applications. As Behl & Raj, (2018) further elaborate, PWAs emerged as the optimal solution to provide the user with a native app-like experience and offer the advantages of web applications. The most notable platforms using the PWAs today are social media platforms through their lite Versions (Twitter Lite, Facebook Lite, and Instagram Lite).

References

Behl, K., & Raj, G. (2018). Architectural Pattern of Progressive Web and Background Synchronization. *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 366–371. https://doi.org/10.1109/ICACCE.2018.8441701

Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T.-M. (2018). Progressive Web Apps for the Unified Development of Mobile Applications. *International Conference on Web Information Systems and Technologies* (pp. 64–86). Springer. https://doi.org/10.1007/978-3-319-93527-0_4

MDN Contributors. (2021). *Progressive web apps (PWAs) | MDN*. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

Steiner, T. (2018). What is in a Web View. *Companion on The Web Conference 2018 - WWW '18*, 789–796. https://doi.org/10.1145/3184558.3188742