INFO3180 - LECTURE 11

PROGRESSIVE WEB APPS (PWA)

A Progressive Web App (PWA) is a traditional web app that is progressively enhanced, using open web technologies, to help it deliver the best possible experience on every device (based on its available capabilities).

Source: https://dev.to/azure/01-introducing-progressive-web-apps-hi4

WHAT DO WE MEAN BY PROGRESSIVE ENHANCEMENT?

- Developers prioritize a baseline experience where users can consume core content from any device, even those with older browsers.
- Developers then detect support for newer features on devices to unlock enhanced experiences on these more-capable platforms.

Source: https://dev.to/azure/01-introducing-progressive-web-apps-hi4

NATIVE APPS VS PROGRESSIVE WEB APPS

- Native apps work only on their target device platforms, requiring a dedicated codebase (and team) for each target, but unlocking richer hardware-enabled features.
- Web apps work everywhere with a single codebase (and team) - but are limited in their ability to use platform-specific features to ensure consistent experiences across platforms.

Source: https://dev.to/azure/01-introducing-progressive-web-apps-hi4

NATIVE APPS VS PROGRESSIVE WEB APPS CONT'D

- Typically native apps are considered to be more performant depending on what type of app you are buidling
- Web apps can be more cost effective since you can use the same team and technologies across platforms.

Source: https://dev.to/azure/01-introducing-progressive-web-apps-hi4

CHARACTERISTICS OF PWA'S

- **Discoverable** - Because a progressive web app is a website, it should be discoverable in search engines.
- **Installable** - can be installed on a devices home screen so it can be launched easily
- **Linkable** - do your best to ensure users can save their place by bookmarking it or when they re-launch their web browser and your site’s tab is re-launched.
- **Network Independent** - should work in areas of low connectivity or offline. This is where a Service Worker will come in.

CHARACTERISTICS OF PWA'S

- **Progressive** - Works for every user, regardless of browser choice because it's built with progressive enhancement as a core tenet.
- **Re-engageable** - you can use 'push notifications' to re-engage your users when new content is available.
- **Responsive** - should fit the device’s form factor and screen size. Whether desktop, mobile, tablet, or whatever is next. This is where Responsive Web Design comes in.
- **Safe/Secure** - it is required that the app be hosted over HTTPS

CHARACTERISTICS OF PWA'S

- **App-like** - it should look like a native app and be built on the application shell model, with minimal page refreshes.
- **Fresh** - Always up-to-date thanks to the service worker update process.

At the very least your PWA should be running under HTTPS, have a Web App Manifest and utilize a Service Worker.

WHAT IS A WEB APP MANIFEST?

A Web App Manifest provides information about an application (such as name, author, icon, and description) in a JSON text file. The purpose of the manifest is to install web applications to the homescreen of a device, providing users with quicker access and a richer experience.

EXAMPLE APP MANIFEST

```
{
  "short_name": "Flask PWA",
  "name": "Demo Flask Progressive Web App (PWA)",
  "icons": [
    {
      "src": "launcher-icon-1x.png",
      "type": "image/png",
      "sizes": "48x48"
    },
    {
      "src": "launcher-icon-4x.png",
      "type": "image/png",
      "sizes": "192x192"
    }
  ],
  "start_url": "index.html",
  "display": "standalone",
  "theme_color": "#ffcc66",
  "background_color": "#ffffff",
}
```

TELL THE BROWSER ABOUT YOUR MANIFEST FILE

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

Place this in the **<head></head>** of your HTML document.

OFFLINE ACCESS WITH SERVICE WORKERS

It's becoming increasingly important for our web apps to still function (to some degree) even if a network connection becomes unavailable.

HTML5 introduced a number of features to help make our web apps work offline. Some of these features are Web Storage, application cache, Service Workers, IndexedDB, CacheStorage, online and offline events, to name a few.

We have already looked at Web Storage (localStorage and sessionStorage) and today we will focus on Service Workers and online and offline events.

Service workers essentially act as proxy servers that sit between web applications, and the browser and the network (when available).

They are intended to (amongst other things) enable the creation of effective offline experiences, intercepting network requests, and taking appropriate action based on whether the network is available, and updated assets reside on the server. They will also allow access to push notifications and background sync APIs.

ADVANTAGES OF SERVICE WORKERS

Service Workers gives an application three advantages:

- Offline browsing - users can use the application when they're offline
- Speed - cached resources load faster
- Reduced server load - the browser will only download updated/changed resources from the server

So how do we create and activate a service worker?

REGISTERING A SERVICE WORKER

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker
    .register('/service-worker.js')
    .then(function() {
      console.log('Service Worker Registered');
    });
}
```

You need to ensure you that register your service worker.

CACHE OUR SITE ASSETS ON FIRST VISIT

```
let cacheName = 'pwa-cache-v1';
let filesToCache = ['/', 'index.html', 'app.js',
'styles.css', 'logo.png'];

self.addEventListener('install', (e) => {
  // use the install event to pre-cache initial resources
  e.waitUntil((async () => {
    const cache = await caches.open(cacheName);
    await cache.addAll(filesToCache);
  })());
});
```

This is located in your service-worker.js file.

ACTIVATE SERVICE WORKER AND UPDATE THE CACHE

```
self.addEventListener('activate', function(e) {
  e.waitUntil((async () => {
    const cacheKeys = await caches.keys();
    cacheKeys.map(async (key) => {
      if (key !== cacheName) {
        // Removing old cache
        await caches.delete(key);
      }
    });
  })());
  return self.clients.claim();
});
```

This is located in your service-worker.js file.

FETCHING CACHED ASSETS AND NETWORK ASSETS

```
self.addEventListener('fetch', async function(e) {
  e.respondWith((async () => {
    const cache = await caches.open(cacheName);
    const cachedResponse = await cache.match(e.request.url);
    if (cachedResponse) { return cachedResponse; }

    try {
      const fetchResponse = await fetch(e.request.url);
      return fetchResponse;
    } catch(error) {
      // Fetch failed; returning offline page instead.
      const cachedResponse = await cache.match('offline.html');
      return cachedResponse;
    }
  })());
});
```

This is located in your service-worker.js file.

SOME CACHING STRATEGIES

- **Network or Cache -** retrieve the most up to date content from the network but if the network is taking too long, it will serve cached content instead.
- **Cache Only -** always retrieve content from the cache whether or not you are online.
- **Cache and update** - respond from cache to deliver fast responses and also updating the cache entry from the network.
- **Cache, update and refresh -** respond from cache to deliver fast responses and also update the cache entry from the network. When the network response is ready, the UI updates automatically.

Workbox is a set of libraries to help you write and manage service workers and caching with the CacheStorage API.

https://developer.chrome.com/docs/workbox/

Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, and more.

https://developers.google.com/web/tools/lighthouse/#devtools

You can also check if a user is online or offline. HTML5 introduced the navigator.onLine property.

In addition, you can be notified of any connectivity changes by listening to the online and offline events of the window element.

```
if (navigator.onLine) {
    // send something to the server
} else {
    // save to localStorage
}
```

```
window.addEventListener("online", function() {
  // send any changes that were pending when offline to the
server
}, true);

window.addEventListener("offline", function() {
  // display a message and perhaps save to localStorage
  // until the user is back online then send to server
  alert("You're now offline. If you update your status, it
will be sent when you go back online");
}, true);
```

Some other features you may want to look into with PWA's are Push Notifications and Background Sync.

PUSH NOTIFICATIONS AND BACKGROUND SYNC

- Web push notifications allow users to opt-in to timely updates from sites they love and allow you to effectively re-engage them with customized, relevant content. The Push API and Notification API open a whole new set of possibilities for you to re-engage with your users.
- Background sync is a new web API that lets you defer actions until the user has stable connectivity. This is useful for ensuring that whatever the user wants to send, is actually sent once connectivity is re-established.

Lastly, "literally any website can be a progressive web app. If you work on a website, there's a really good chance your site could benefit from the technologies and approaches aggregated under the PWA umbrella."

Source: https://www.aaron-gustafson.com/notebook/your-site-should-be-a-pwa/

RESOURCES

- 30 Days of PWA's - https://dev.to/azure/01-introducing-progressive-web-apps-hi4
- MDN Progressive Web Apps - https://developer.mozilla.org/en-US/Apps/Progressive
- Web.dev PWA - https://web.dev/progressive-web-apps/
- A Beginner's Guide to PWA's - https://www.smashingmagazine.com/2016/08/a-beginners-guide-to-progressive-web-apps/
- Yes, That Web Project Should be a PWA - https://alistapart.com/article/yes-that-web-project-should-be-a-pwa

RESOURCES

- Service Worker API - https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API
- Service Worker Cookbook - https://serviceworke.rs/
- PWA Checklist - https://web.dev/pwa-checklist/
- Lighthouse - https://developers.google.com/web/tools/lighthouse/#devtools
- Workbox - https://developer.chrome.com/docs/workbox/
- Progressive Enhancement - https://alistapart.com/article/understandingprogressiveenhancement

RESOURCES

- PWA Builder - https://www.pwabuilder.com/
- Background Sync - https://developers.google.com/web/updates/2015/12/background-sync
- Push Notifications - https://developers.google.com/web/fundamentals/push-notifications/

DEMO

https://github.com/uwi-info3180/flask-pwa