# Progressive Webapp

**Definition:**

Any application that delivers app-like experience in the browser and can be developed progressively (Can be developed in layers that are independent of each other)

**Characteristics:**

1. Instant loading 7. Linkable
2. Discoverable 8. Re-engageable(Push notification)
3. N/W independent 9. Works everywhere
4. Responsive 10. Fast
5. Installable
6. Secure

**Progressive Enhancement: Adding features incrementally in layers.**

1. It’s a website
2. Adds native installation
3. Adds Web push notifications
4. Add hardware & platform access

**PWAs and Hybrids:**

1. Big differences
2. Store distribution – Need to pay for Hybrid apps and need to package them
3. No packaging and signing for PWAs
4. No native plugins for PWA. This is where Hybrids take the lead**.**

**PWAs compatibility:** <http://gs.statcounter.com/>(Site for getting stats for browser usage)

1. Chrome on Android
2. Opera Mobile
3. Samsung Internet browser (Chromium based)
4. Firefox developer edition
5. Edge (future, IE +16)
6. Limited APIs available for iOS, so limited support. iOS 11 may be better performing in PWAs
7. Chrome iOS is not chrome. Check user-agent string for chrome.
8. Each instance of the PWA depends on the browser engine it has been installed. Eg: If we installed the same app from 2 different browsers, then we would have 2 separate instances of the app running independently. This is a problem to be addressed.
9. We could save the credentials of the user if there is only one engine that the app is being installed. Currently this is supported only in Android. But there is a credentials management API in the pipeline that helps the OS manage the user session.
10. PWA is ideal for the apps that are single page applications.
11. PWA supports web-sockets as the browsers supporting the PWAs do that.
12. PWA is great asset to the Physical web. (https://google.github.io/physical-web/)

**Limitations:**

1. Multiplatform support, eg: camera might different in different platform.
2. Responsive design – Taking into account network, UI rendering and other cases for better design.
3. PWAs do not get the privileges of First class citizen like the native apps.
4. Intents
5. Distribution, PWAs are not present in the store.
6. Social networking sites cannot be acting as a discovery mechanism for PWAs. The reason is that a lot of the time in the mobile devices the social networking apps open their own internal browser. These do not support PWAs.
7. Web app installers also could be ineffective as the user has to visit the website twice for getting the PWA install option.

Web App manifest: Eg: <https://www.flipkart.com/manifest.json> (Change to device mode in chrome debug tool)

* Should be based on W3C spec.
* It is a JSON file.
* Should be served as of type application/manifest.json
* name: The name shown on the splash screen.
* Start\_name: The name shown under the icon on the mobile home page.
* Start\_url : The url from where the app should or opened.
* Orientation : any, portrait [-primary | -secondary], landscape [-primary | -secondary], natural(Lets the device decide what is best suitable.)
* Display:
  + Browser : Not actually PWA
  + minimal-ui : Lite version of the app. Browser UI but with little options to edit the options say URL etc. Currently similar as standalone
  + **standalone**: Most web-apps use this for mobile app experience.
  + fullscreen: Mostly games where we miss the battery status, top bar of phone.
* background\_color:
* theme\_color: The color of the theme for the app.
* Icon: array of the icons with props
  + Src
  + Sizes: size of the icons for diff platforms.
  + Type

**Service workers:**

* One of the two important core APIs for the PWA apart from manifest.json
* It is a web worker
* It has one scope and abilities over that scope.
* It works detached from any tab on the browser. Has its own lifecycle.
  + **Web Worker**
    - A JS thread.
    - No access to UI or DOM
    - No document , no window.
    - Separate JS file.
  + **Scope**
    - Just an origin(protocol + host (domain) + port) and path
    - Default and Max scope
    - Can be overridden with “Service-Worker-Allowed” flag.
    - Eg: Scope: https://fluent.com/files
* Service worker can see all the resources that pages on the scope request, even cross-origin.
* It can respond for those resources synthesizing responses or fetching them. Acts like a proxy on client side.
* Service worker is independent of the website or the Tab. It has its own lifecycle. We can even debug the code in the service worker.
* Each website can have its service worker with a scope.
* Service worker can send messages to clients using the client connection API.
* The only way to have the browser check for new version of Service worker is to have no-cache tag in the HTTP header in the js file for the service worker.
* The service worker has a cache. The cache stores key value pairs of the HTTP request and response.
* The service worker cannot access any synchronous API for Eg: Local storage API.
* The service worker must be served only using TLS (HTTPS), localhost exception.
* Service worker is based on promises.
* The service worker must be in the root folder if it must access all the resources under the domain.
* Each service worker file will be installed only once.
* It is safer to use arrow functions within a service worker
* <https://github.com/GoogleChrome/sw-precache>
* <https://github.com/GoogleChrome/workbox> - Collection of libraries for Service worker.
* Clients API for SW to access the client. It is a part of the PWA running in the browser.
  + claim().then()
  + openWindow()
  + matchAll(options).then
    - includeuncontrolled
  + focus()
  + navigate(url).then()
  + Many other options.
* navigator.onLine = false means we are offline. If true then the OS is telling that we are online. But we cannot tell if we are connected to the internet. The best way is to try to ping to the server.

**Use cases of Service worker:**

* Offline web. Similar to an Apache Server. Service worker would act as an proxy that handles the logic at the client end.
* PWA can be effectively built using service worker.
* Deliver assets immediately from cache. Or provide local resources faster to the user.
* Background update of resources and assets asynchronously.
* React to bad n/w connections. If no updated resources is returned then render the cached version.
* Better response to non-200 responses.
* Prefetching resources and assets before time.
* Local content generation.

Notes for iOS:

* <meta name="apple-mobile-web-app-capable" content="yes"> add this for iOS devices.
* In iOS the PWAs are not safari processes. They are stored as process with extension “app”. iOS 11 would have better support.
* <meta name="apple-mobile-web-app-title" content="Fluent"> to add a name for the icon.
* <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent"> along with the header tag and css styles can be used to emulate the theme property of manifest.json file in iOS
* In iOS the PWAs are reloaded every time we open the app.
* <link rel="apple-touch-icon" href="icon\_ios.png"> to add the icon for PWA in iOS devices like iPhone and iPad.

## **Links**

Web App Manifest Generators

<https://tomitm.github.io/appmanifest/>

<https://app-manifest.firebaseapp.com/>

<https://brucelawson.github.io/manifest/>

Icon Generators

<http://appicon.build/>

[https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html](https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html#foreground.type=clipart&foreground.clipart=android&foreground.space.trim=1&foreground.space.pad=0.25&foreColor=rgba(96%2C%20125%2C%20139%2C%200)&backColor=rgb(68%2C%20138%2C%20255)&crop=0&backgroundShape=square&effects=none&name=ic_launcher)

## **Snippets**

### **iOS Meta Tags**

<meta name='viewport' content='width=device-width,initial-scale=1'>

<meta name="apple-mobile-web-app-capable" content="yes">

<meta name="apple-mobile-web-app-title" content="Fluent 2017">

<meta name="apple-mobile-web-app-status-bar-style"

     content="black-translucent">

<link rel="apple-touch-icon" href="icon\_ios.png">

### **Service Workers Snippets**

**Registration**

if ('serviceWorker' in navigator) {

   navigator.serviceWorker.register("serviceworker.js")

       .then(function(registration) {

   // Worker is registered

   }).catch(function(error) {

   // There was an error registering the SW

   });

}

**Basic Add to Cache**

var urls = [];

self.addEventListener("install", function(event) {

   console.log("The SW is now installed");

   event.waitUntil(caches.open("myAppCache").then(function(cache) {

       return cache.addAll(urls);

   }));

});

**Fetch**

self.addEventListener("fetch", function(event) {

   event.respondWith(caches.match(event.request)

       .then(function(response) {

           if (response) {

               // The request is in the cache

               return response;

           } else {

               // We need to go to the network

               return fetch(event.request);

           }

       })

   );

});

**Fetch: Stale while Revalidate**

self.addEventListener('fetch', function(event) {

   event.respondWith(

       caches.match(event.request)

           .then(function(response) {

               // Even if the response is in the cache, we fetch it

               // and update the cache for future usage

               var fetchPromise = fetch(event.request).then(

                   function(networkResponse) {

                       caches.put(event.request, networkResponse.clone());

                       return networkResponse;

                   });

               // We use the currently cached version if it's there

               return response || fetchPromise;

           })

       );

   });

**Deletion**

self.addEventListener('activate', function(event) {

 // Array of cache that we will use in this version

 var cacheWhitelist = ['pages-cache-v1', 'blog-posts-cache-v1'];

 event.waitUntil(

   caches.keys().then(function(cacheNames) {

     return Promise.all(

       cacheNames.map(function(cacheName) {

         if (cacheWhitelist.indexOf(cacheName) === -1) {

           // Deletes the cache because we won't use it here

           return caches.delete(cacheName);

         }

       })

     );

   })

 );

});