**Progressive Web App Step-By-Step Course**

**‘Progressively Enhanced Websites’**

**Craig West**

[**Resource List**](#resources)

**Repo**

<https://github.com/iwswordpress/pwa>

**Online Site** <https://wp-html.co.uk/courses/pwa>

*Use lesson number as required e.g. …/pwa/20*

<https://www.youtube.com/watch?v=k_lHvNL0gkw>

**Best learning tools:**

Academind PWA $15USD (Udemy has regular sales) Udemy.com VIDEO

<https://www.linkedin.com/learning/vanilla-javascript-service-workers/> $40USD VIDEO

Building Progressive Web Apps by Tal Ater, O’REILLY Publishers $20USD BOOK

<https://developers.google.com/web/ilt/pwa/> GOOGE FREE VIDEOS/SITE

<https://serviceworke.rs/>

**YouTube Net Ninja PWA** <https://www.youtube.com/watch?v=4XT23X0Fjfk&list=PL4cUxeGkcC9gTxqJBcDmoi5Q2pzDusSL7> FREE

<https://frontendmasters.com/> Variety of courses on PWA, Service Workers…

**Google Project Fugu – to make PWAs the same as a Native App**

<https://docs.google.com/spreadsheets/d/1de0ZYDOcafNXXwMcg4EZhT0346QM-QFvZfoD8ZffHeA/edit#gid=557099940>

**Especially useful resources we will be using:**

This has page on what PWAs are capable of today and with level of browser support and code snippets:

<https://whatwebcando.today/>

<https://serviceworke.rs/>

<https://jakearchibald.com/2014/offline-cookbook/#generic-fallback>

<https://developers.google.com/web/ilt/pwa/>

**RESOURCE LIST at end of document.**

**This workshop covers 4 areas:**

1. Creating and installable app.

2. Using Service Workers and the Cache API to store pages on the device.

3. Using IndexedDB, an asynchronous key:value pair database.

4. Using BackgroundSync to save posted data when offline and send to server when back on line.

DEMO of app we are building…

**10s UI Building a Native App look**

**10**

**Initial Setup**

This contains the basic index.html before anything is done to make a PWA.

<https://whatwebcando.today/> is a link on the page. It will show you what APIs are available for your browser. One can also click on API to see how to use it. I will demo…

Go through DEV tools

Ensure ‘Disable cache’ on Network Tab of Dev tools.

DEV tools TABS can be rearranged.

Run Lighthouse (RL)

<https://developers.google.com/web/progressive-web-apps/checklist?utm_source=lighthouse&utm_medium=devtools>

Google checklist of banner install:

<https://developers.google.com/web/fundamentals/app-install-banners/native>

**11 Add manifest file**

**What is a manifest file?**

<https://pwa-workshop.js.org/1-manifest/#manifest-fields>

**TOOLS TO CREATE MANIFEST FILE**

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

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

<https://www.pwabuilder.com/>

This has:

* PWA icons added in folder images-manifest.
* Link to manifest file.
* manifest.json is popular use but originally and still useable is manifest.webmanifest
* We only need to add on one page as app will work in this way.

*We can view the manifest file in DEV tool…*

**12 Adding more UI details**

* Added <meta name="theme-color" content="#2196f3" />
* Added Apple and Microsoft UI

Run Lighthouse again.

**13 Register a SW**

Add Service Worker.

Only need to install once.

I will know give some theory about SW location…

**14 Add FETCH event listener to enable Add to Homescreen**

Add lifecycles to SW to enable ADD to HOMESCREEN.

Lighthouse requires a SW with a FETCH event (even if empty) to qualify as PWA.

Google checklist of banner install:

<https://developers.google.com/web/fundamentals/app-install-banners/native>

Should see this when you go to <https://wp-html.co.uk/courses/pwa/14>

A screenshot of a cell phone

Description automatically generated

Desktop URL bar has + and a slide in.

A close up of a logo

Description automatically generated

**15**

We add a button to activate the A2HS banner and a script to capture the install event and to fire later.

We can also prevent the mini-info bar buy using **e.preventDefault()** in the **beforeinstallprompt** event where we can save the event for future calling.

If the custom prompt is rejected, then there may be a time limit for when it can reappear.

<https://developers.google.com/web/updates/2018/06/a2hs-updates>

<https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Add_to_home_screen>

A screenshot of a cell phone

Description automatically generated

**This completes the UI part of building a PWA.**

**20s Cache API and Service Workers**

<https://docs.google.com/presentation/d/1bFVDRa71JK0ilr6lS6y9Twqp9gy8jzV1O8FFbG4A3qk/edit#slide=id.p9> (Google Slides)

<https://developers.google.com/web/fundamentals/primers/service-workers>

**Some SW Theory**

<https://mydomain.com/>

<https://mydomain.com/myapp>

<https://mydomain.com:8500/>

<https://subdomain.mydomain.com/myapp>

* all different scopes – **scheme, domain, port make up the origin.**
* One SW can point to many clients (iframe/page/pwa).
* SW has no access to page DOM for security reasons.
* Acts as a ‘proxy’ between website and os but is in site folder.
* Any page can register a SW – no permission needed.
* We can have many SW but must have different scopes.
* SW can see all network requests in scope even cross domain.
* Can go to cache, Network or create a new Response.
* Works even when browser closed as it is part of system not site.
* Has many events like SYNC, FETCH, PUSH, MESSAGE, PAYMENT API
* SW is mandatory for PWAs.
* HTTPS unless localhost.
* Only one file per ‘dominion’ – may import scripts. One entry point.

**20s**

**SERVICE WORKER REFENCES:**

<https://serviceworke.rs/>

<https://jakearchibald.com/2014/offline-cookbook/#generic-fallback>

<https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker>

**20 Installing a SW**

Scope option not used usually as the SW is kept in root of folder SW will have control over.

Install only occurs once so when we refresh page it is not reinstalled.

If we change SW by one byte a new SW is installed but kept in waiting.

A screenshot of a cell phone

Description automatically generated

Note how we can see how many clients (3) we have.

We can import files into SW if we want to break up our code:

Index02.html

*console*.log("We are a service worker");

try {

    importScripts("events.js");

} catch (e) {

}

*console*.log("after loading the events.js script");

**All requests still go through browser’s own cache and are subject to its caching rules.** <https://jakearchibald.com/2016/caching-best-practices/>

As the browser checks for new SW once caching time expires, we can programmatically run the checking algorithm without checking caches.

ADVANCED

Some browsers will only check main SW not imported scripts inside.

{ updateViaCache: “imports|all|none”} in registration

navigator.serviceWorker.register(

'./sw.js',

{scope: '.'},

{updateViaCache: 'imports|none|all'}

}

Imports – default cache only for import, sw from network.

All – use cache for all files.

None – never use cache but go to network.

**21 Storing pages in the CACHE API**

<https://developer.mozilla.org/en-US/docs/Web/API/Cache>

**Preamble**

Let us think of the Cache API as our local web server where we store our web pages. It is a cache that our SW can manage and is NOT the browser’s cache. All request will still go through the browser cache and be subject to its caching policies

We can intercept network requests and implement various strategies:

Cache only

Network only

Cache first else Network

Network first else cache

Cache first then also NetworkThnStoreInCacheForNextTime (‘stale while revalidate).

Generic fallback page when all the above fail. We can parse the request and implement different responses accordingly.

We can also parse the request and apply different strategies e.g. **network only** for stock day traders and **cache else network** for stylesheets

We would probably want to preload the app shell (**precache**) and then add other files as they are requested, (**dynamic caching**).

As caching is Promised based meaning that if one request fails the whole batch fails, we can precache our pages in one cache and have remote resources like fonts etc cached in another cache.

We now pre-cache the APP-SHELL.

We use some constants for the two local caches.

If we have a typo or a missing file, then the service worker will not load the assets as it is transactional effectively – all success or not at all.

As we may have issues to remote assets, we separate local and remote assets.

Load dev tools and load page. You will see the Cache get an arrow and then there will be two caches.

NB We need to add ‘./’ for when full path not specified. The absolute dot seems to be needed as we are not in root of site.

A screenshot of a social media post

Description automatically generated

As a test, make a typo and see how local cache is empty. Use Clear Storage on Application tab to clear out caches and do a hard refresh.

Go OFFLINE in the Network tab – we still get the dinosaur as we have not yet intercepted the request and respond to it. We have just cached pages.

We now add code for the fetch event to intercept requests and respond with the cache if present.

Ensure Disable Cache is checked and with the new code set to offline in Network tab.

Reload (soft) and the page remains.

HARD RELOAD bypasses SW (BYPASS FOR NETWORK checkbox in DEV TOOLS > APPLICATION > Service Workers

In the screenshot below, you can see files being fetched by Service Worker and those that have not been cached failing as we have no network connection.

We will now add all the APP-SHELL files as well as the fallback.html file needed to return a graceful offline page if there the requested page is not in the cache.

A screenshot of a cell phone

Description automatically generated

**22 Going offline**

This has all the necessary pre-cache files for the rest of the course added to the sw.js.

We have added four sample.html pages to see the fallback page in operation. Form.html is also not precached.

A screenshot of a social media post

Description automatically generated

**23 Storing new pages in the DYNAMIC CACHE and generic FALLBACK\_PAGE**

We have now enabled offline use for the APP-SHELL, index page and a generic fallback page.

What about pages that have not been cached?

We could cache all pages on the site, but this would slow down initial load.

We can enhance the fetch event in the SW by creating a DYNAMIC CACHE that store pages that are returned from the network.

If the page is not in the cache, we continue to network, get the page and clone a copy of it (it is a stream – I will explain) and store it in the DYNAMIC CACHE and return the page. Then next time we can retrieve it from this cache.

The sw.js now has a variable for the DYNAMIC CACHE and from now on we will use different SW files as needed for our caching strategies.

Reload site and we still have the two PRECACHE.

If we now request sample01.html whilst ONLINE, we will see it being added to the newly created DYNAMIC CACHE.

A screenshot of a social media post

Description automatically generated

Any files that it gets that are not in the PRECACHE are loaded in.

Right-click and delete this cache, reload (soft) the page and see it being recreated.

Get sample02.html then sample03.html and sample04.html. You will see it store all the pages and images.

There are two JS files that use the browser’s own **navigator.online** API.

<https://developer.mozilla.org/en-US/docs/Web/API/NavigatorOnLine/onLine>

**gone-offline.js** responds to a change in online|offline status on a page.

**style-offline.js** detects online|offline status and changes background to #ccc for offline and #fff for online.

(both stored in \_js/ folder)

Go offline and refresh page.

These pages will now be available offline.

There is one problem. If we change version number, we keep adding caches. The SW cannot tell what one is the latest by its name and the order in the caches may not be the order it looks in.

Thus, when we change the version number in SW we need to clear out old cache versions.

This is best done in ACTIVATE event and the code is initially commented out. If we uncomment it, we will see that only the latest version remains.

This filtering can be done on the generic fallback section where we only return the fallback.html if the request is for an html page rather than a failed return of an image, which could have its own fallback.jpg.

From <https://jakearchibald.com/2014/offline-cookbook/#generic-fallback>

**24 Purging old caches and limiting size of dynamic cache**

We now purge and limit the size of the dynamic cache.

**Summary**

We now have a PWA that has the UI of native apps and the offline capability with cache management as well.

We have precached the APP SHELL for fast load and have a graceful FALLBACK page for pages not found whilst offline.

If we visit a page that has not been cached, it is cached for future use.

There is Google Workbox (npm and cdn) that can be used to configure service worker strategies:

<https://developers.google.com/web/tools/workbox>

**30s IndexedDB**

IndexedDB was developed around 10 years ago and is based on callbacks and can seem complex to use.

There are new libraries that make use of Promises and these are listed below.

* IDB is a key-value pair database and not relational.
* It uses Javascript objects so can store files, images and videos.
* It is transactional – if one action fails they all fail.
* It has a same-origin policy meaning sites can only read their own IDB.

In our APP we use Jake Archibald’s IDB-Promised library (idb.js) and we have some CRUD functions in idb-fn.js.

<https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB>

There are several libraries and we will use a few of them (in red):

IDB-Keyval: <https://github.com/jakearchibald/idb-keyval>

IDB-Promised: <https://github.com/jakearchibald/idb-keyval>

DexieJS: <https://dexie.org/>

LocalForage: <https://github.com/localForage/localForage>

There is also JSStore(<https://jsstore.net/>), that can be used. Along with DexieJS, they can perform SQL type operations and queries.

**30 Load and store data in IndexedDB**

Open idb.html

In Dev tools > Application>IndexedDB you should see:

A screenshot of a social media post

Description automatically generated

As well as:

A screenshot of a social media post

Description automatically generated

Load this page and then go to POSTS.

Check out IndexedDB.

We will move all the IndexedDB loading into the HOME page in the next lesson.

dexie-read.html and dexie-write.html uses the DexieJS library. They load contacts from randomuser.me and once loaded the CONTACTS on the menu will show them. This page will work offline too but load it first whilst online as it has not been precached.

Do the same for articles.html and article.html which will display the WP POSTS.

If we now add these files to the PRECACHE, we can then enter site, go offline and all the tables will work.

**31 Move IDB load to index.html. Use dexie.js.**

This has all files precached.

Index.html has scripts to load POSTS (idb.js) from WordPress and CONTACT (dexie.js) data from randomuser.me api.

You may need to clear storage, and close and reopen site.

The PWA should now enable you to view POSTS and CONTACTS even if you go offline once on HOME page. You will not need to open the other pages as the data will have been loaded on HOME page.

Note how in Network Tab the files come from the (ServiceWorker):

A screenshot of a cell phone

Description automatically generated

This is the Local Storage part of the course completed.

We are now able to store data in IndexedDB – a sort of MySQL – and with template pages we can access data. The template pages are precached enabling rendering even if the page has never been visited before connectivity is lost.

At this stage we have an immensely powerful PWA that uses not just the Cache API but also IndexedDB.

We have in effect a local dev server on our device combined with Native App UI and Add to Homescreen.

We can now go to HOME and all the data will be loaded. POSTS and CONTACTS will work offline.

We have not loaded FORM (intentionally) so if OFFLINE and we go to FROM we will get the FALLBACK page.

Once ONLINE and we go to FORM, it will be added to the DYNAMIC CACHE for offline use.

**40s Offline POST**

**40**

**Look at notifications.html to see how the Notification API works in regular pages.**

This has the ENABLE NOTIFICATIONS button added.

Click on the button and see the custom notification.

In URL bar on left before IP address click on the I and see how you can reset the permission status.

Blocked prevents the dialog request box, allow grants it and will send notification on clicking the. If ‘granted’ button is disabled and if user changes via setting it forces a reload and button will become enabled again.

We will now cover some theory.

Better described as BackOnlineSync, the SW listens for when the browser has connectivity again and then carries out some work.

When a user submits a form and is offline, we do the following:

Register a BackgroundSync event with a name like ‘SendData’

Save form data in local storage.

Create an event in SW that listens for BackgroundSync and then carries out the BackgroundSync event ‘SendData’ we set up.

A notification can be sent to user once this is done.

Please note the browser does not need to be open. It is a OS level activity like badge notifications for apps like email and FB.

Ensure Allow Notifications is setting (left UTL bar).

Fill out form online – view localForage and console

Fill out offline.

**41**

Ensure NOTIFICATIONS is enabled.

FORM page.

Enter one field is sufficient to SEND.

Notice the NOTIFICATION.

In DEV TOOLS, you can see the SYNC event that gets fired.

The SYNC event is ‘back online’ event, so if online it carries out instructions.

GO OFFLINE and send form.

Note no NOTIFICATION and in DEV TOOLS see the SYNC EVENT is not there.

GO ONLINE and NOTIFICATION appears and SYNC event will be in CONSOLE.

On mobile, this happens if the browser is closed. (Send form in FLIGHT MODE, close app and then go back online).

On desktop, if the browser is closed and then opened up again, this will replicate the FLIGHT MODE -> ONLINE of mobile.

**42**

We add a message sent from SW to PAGE

    self.clients.matchAll()

      .then(

*function* (*clients*) {

          clients.forEach(*function* (*client*) {

*console*.log(client);

            client.postMessage("- - - postMessage from SW...form sent")

          })

        }

      )

**Messaging between clients and SW**

**50**

Example code of page button sending message to SW and then getting a reply from SW.

Progressive Web Apps – Resource Sheet

[craig@wpjs.co.uk](mailto:craig@wpjs.co.uk)

**TO LEARN PWA**

This has page on what PWAs are capable of today: <https://whatwebcando.today/>

*Many free courses:*

Net Ninja PWA on YouTube is an excellent set of videos: <https://www.youtube.com/watch?v=4XT23X0Fjfk&list=PL4cUxeGkcC9gTxqJBcDmoi5Q2pzDusSL7>

Google has a great set of code labs, videos, slide shows.

**This playlist on YouTube has 30 plus lessons:**

<https://www.youtube.com/watch?v=psB_Pjwhbxo&list=PLNYkxOF6rcIB2xHBZ7opgc2Mv009X87Hh>

<https://jakearchibald.com/2014/offline-cookbook/>

<https://developers.google.com/web/ilt/pwa>

<https://developers.google.com/web/fundamentals/codelabs>

<https://codelabs.developers.google.com/dev-pwa-training/>

<https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/>

Jake Archibald at Google has created many resources so a search for his name yields great results.

Udacity has a few free courses built by Google.

Two good free courses:

1. <https://pwa-workshop.js.org/>
2. <https://pwa-cookbook.js.org/#/a-propos>

Best book (excellent) is Tal Ater’s Progressive Web Apps:

<https://www.amazon.co.uk/Building-Progressive-Web-Apps-Ater/dp/1491961651>

Best PAID for PWA course (£15) is <https://www.academind.com/>

**TOOLS TO CREATE MANIFEST FILE**

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

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

<https://www.pwabuilder.com/>

**CODE EXAMPLE SITES**

Mozilla: <https://serviceworke.rs/>

Jake Archibald’s Offline Cookbook: <https://jakearchibald.com/2014/offline-cookbook/>

Push Notifications: <https://web-push-book.gauntface.com/demos/notification-examples/>

**LATEST CHROME DEV SUMMIT VIDEOS**

Google CDS Nov 2019 on their intent to close gap between Native Apps and PWAs:

<https://www.youtube.com/watch?v=JKVZMqpiY7w&t=12s>

They have a spreadsheet showing progress and plans for future APIs:

<https://docs.google.com/spreadsheets/d/1de0ZYDOcafNXXwMcg4EZhT0346QM-QFvZfoD8ZffHeA/edit#gid=557099940>

**OTHER**

Is Service Worker Ready? <https://jakearchibald.github.io/isserviceworkerready/resources.html>

Storage Limits: <https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa>

App Install Requirements: <https://developers.google.com/web/fundamentals/app-install-banners>