TURNING MOODLE WEB INTO A PROGRESSIVE WEB APP (PWA)

June 2018

Mitxel Moriana
Developer @ 3iPunt Moodle Partner
WHAT IS A PWA?

Web App stands for any web app, like Moodle!

P stands for progressive or “optionally enhanced”:

if browser supports this cool feature {
    use it
} else {
    // no problem! Do nothing
}
TYPICAL EXAMPLES OF “PROGRESSIVENESS”

if ('serviceWorker' in navigator) {
    register our Service Worker
}

if ('serviceWorker' in navigator && 'PushManager' in window) {
    we can use the Push API in our Service Worker
}

if ('caches' in window) {
    we can use the CacheStorage API to access storages/caches (i.e.: key-value pairs, request-responses pairs) shared by the window and the Service Worker
}
THE SERVICE WORKER (I)

Not be confused with a shared worker

Event-driven script (written in JS) run (when needed) by the browser in the background (i.e.: in its own context/thread, not tied to a page, no direct DOM access).

Allows for the interception (on fetch event) of navigation/requests within its “scope” (usually, but not necessarily, the wwwroot of our site).
THE SERVICE WORKER (II)

Allows for:
• Caching responses (and serving them) at client level
• Serving fallback responses to errors (or any custom condition)
• Pre-caching of responses/resources
• Background data synchronization (periodic data sync is “experimental”)
• Push notifications

Ideally == when possible == if we do it right;
• Faster navigation
• Offline navigation
• Happier users (also, happier developers)
TURNING MOODLE INTO A PWA
CAUSE STUDY OF ELE.ME (MPA)

Skeleton screen + PRPL pattern
(Preload critical resources, Render initial route, Pre-cache remaining routes, Lazy-load remaining routes).

Results:
• Loading time decreased by 11.6% across all pre-cached pages
• Loading time decreased on average by 6.35% across all pages.
• Time-to-consistently-interactive dropped to 4.93 seconds on a 3G network on first load

More info:
https://h5.ele.me/
https://medium.com/elemefe/upgrading-ele-me-to-progressive-web-app-2a446832e509
TURNING MOODLE INTO A PWA
THE OFFLINE FALLBACK VIEW

Pre-cache an offline view and use it as an offline response fallback (for non-cached view request/responses).
TURNING MOODLE INTO A PWA
CACHING STATIC CONTENT (I)

Caching JS, CSS, images, fonts... when requested at least once.

Moodle serves optimized static content using specific scripts:
- `theme/styles.php` – serves “the one huge CSS of each theme”
- `theme/font.php` – serves “the fonts used in CSS”
- `theme/yui_combo.php` – serves “yui Javascript and CSS”
- `theme/image.php` – serves “the one theme and plugin images”
- `lib/javascript.php` – serves “optimised JS”
- `lib/requirejs.php` – serves “optimised JS for RequireJS”

The URLs to these scripts contain theme/script “versioning” parameters:
New versions => different request URL => no conflict updated vs cached
Choosing the “right” serving and caching strategy/recipe:

Does the URL (including parameters) always returns the same content?

if yes {
  Use “cache only” (if pre-cached!) or “cache first” serving strategies (and dynamically cache them)
}
else if not always, but to load the most recent version IS NOT essential {
  Use “stale while revalidate” serving/caching strategy (serve cached if exists but check in background for an update and cache updated resource when needed)
}
else if not always, but to load the most recent version IS essential {
  Use “network first” serving strategy (and dynamically cache newest version, but use them only as a fallback)
}
else if no and never {
  Are you sure it is static content? Why do we need to cache it?
}
TURNING MOODLE INTO A PWA
CACHING STATIC CONTENT (III)

Lighthouse audit
Mobile emulated, 3G throttled “second visit”, i.e. with browser cache preserved.

With Service Worker serving cached static content:
1260 ms (first meaningful paint)

Without Service Worker:
1430 ms (first meaningful paint)

“First meaningful paint” was 170 ms faster = improvement of ~10% in the “user-perceived loading experience”
TURNING MOODLE INTO A PWA
CACHING STATIC CONTENT (IV)

WITH SW

Performance
55
Progressive Web App
81
Best Practices

Performance
93
These encapsulate your web app's current performance and opportunities to improve it.

Metrics
1.2 s
1.5 s
1.9 s
2.3 s
2.7 s
3.1 s
3.5 s
3.9 s
387 ms
774 ms
1.2 s
1.5 s
1.9 s
2.3 s
2.7 s
3.1 s
3.5 s
3.9 s

First meaningful paint: 1,260 ms
First Interactive (beta): 3,870 ms
Consistently Interactive (beta): 3,870 ms
Perceptual Speed Index: 1,278
Estimated Input Latency: 48 ms

WITHOUT SW

Performance
92
Progressive Web App
81
Best Practices

Performance
55
These encapsulate your web app's current performance and opportunities to improve it.

Metrics
1.2 s
1.5 s
1.9 s
2.3 s
2.7 s
3.1 s
3.5 s
3.9 s
384 ms
787 ms
1.2 s
1.5 s
1.9 s
2.3 s
2.7 s
3.1 s
3.5 s
3.9 s

First meaningful paint: 1,430 ms
First Interactive (beta): 3,940 ms
Consistently Interactive (beta): 3,940 ms
Perceptual Speed Index: 1,366
Estimated Input Latency: 48 ms
TURNING MOODLE INTO A PWA HOME SCREEN & MANIFEST.JSON (I)

manifest.json
{
  app name and short name,  
  icons (icons and splash screens),  
  related applications (web, play store...), 
  start_url (starting url, it could be the root / ), 
  display (standalone = “appish”, browser...), 
  scope (scope url, like for example the root /) 
  background and theme _color (#f98012),
  ...
}
+ <link rel="manifest" href="/manifest.json">  
+ convenient meta tags...  
+ https  
+ ¿use conditions?  
=  
Browser prompts the user to install the home screen  
/ Add to home screen menu option appears
TURNING MOODLE INTO A PWA HOME SCREEN & MANIFEST.JSON (II)

You can set the `manifest.json` in a way that the user can be prompted to install the mobile app from the stores (instead of as a “link” to the PWA)

¿More convenient for customers that have their own Moodle Mobile app?

```json
{
...  
  related_applications: [{ platform: web }, { platform: play, id: com... }]
  prefer_related_applications: true
}
```
TURNING MOODLE INTO A PWA HOME SCREEN & MANIFEST.JSON (III)
TURNING MOODLE INTO A PWA
CACHING PAGE VIEWS (I)

One idea...
Caching the most used views in a given Moodle instance...

Which views are the most “used” in a given Moodle instance?

ANALYTICS can tell us →

<table>
<thead>
<tr>
<th>Path</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>index.php (frontpage), course/view.php</td>
<td>251592</td>
<td>29%</td>
</tr>
<tr>
<td>mod/forum/discuss.php</td>
<td>152132</td>
<td>17%</td>
</tr>
<tr>
<td>mod/forum/view.php</td>
<td>101741</td>
<td>12%</td>
</tr>
<tr>
<td>mod/book/view.php</td>
<td>94548</td>
<td>11%</td>
</tr>
<tr>
<td>mod/page/view.php</td>
<td>45813</td>
<td>5%</td>
</tr>
<tr>
<td>user/view.php, user/profile.php</td>
<td>27685</td>
<td>3%</td>
</tr>
<tr>
<td>mod/book/view.php</td>
<td>19853</td>
<td>2%</td>
</tr>
<tr>
<td>mod/quiz/view.php</td>
<td>17075</td>
<td>2%</td>
</tr>
<tr>
<td>user/index.php</td>
<td>16841</td>
<td>2%</td>
</tr>
<tr>
<td>mod/workshop/view.php</td>
<td>13472</td>
<td>2%</td>
</tr>
<tr>
<td>mod/quiz/attempt.php</td>
<td>13460</td>
<td>2%</td>
</tr>
<tr>
<td>mod/glossary/view.php</td>
<td>11827</td>
<td>1%</td>
</tr>
<tr>
<td>mod/workshop/submission.php</td>
<td>8580</td>
<td>1%</td>
</tr>
<tr>
<td>mod/data/view.php</td>
<td>8549</td>
<td>1%</td>
</tr>
<tr>
<td>mod/quiz/summary.php</td>
<td>7866</td>
<td>1%</td>
</tr>
<tr>
<td>mod/wiki/view.php</td>
<td>6902</td>
<td>1%</td>
</tr>
<tr>
<td>mod/choice/view.php</td>
<td>6784</td>
<td>1%</td>
</tr>
<tr>
<td>mod/quiz/edit.php</td>
<td>4844</td>
<td>1%</td>
</tr>
<tr>
<td>mod/review/view.php</td>
<td>4351</td>
<td>1%</td>
</tr>
<tr>
<td>blog/index.php</td>
<td>4150</td>
<td>0%</td>
</tr>
<tr>
<td>mod/lesson/view.php</td>
<td>4014</td>
<td>0%</td>
</tr>
<tr>
<td>mod/chat/view.php</td>
<td>3636</td>
<td>0%</td>
</tr>
<tr>
<td>mod/wiki/view.php</td>
<td>3606</td>
<td>0%</td>
</tr>
<tr>
<td>mod/lesson/view.php (?)</td>
<td>3573</td>
<td>0%</td>
</tr>
<tr>
<td>mod/chat/report.php</td>
<td>3379</td>
<td>0%</td>
</tr>
<tr>
<td>mod/folder/view.php</td>
<td>3318</td>
<td>0%</td>
</tr>
<tr>
<td>mod/forum/user.php</td>
<td>3028</td>
<td>0%</td>
</tr>
<tr>
<td>mod/assign/view.php (submission page)</td>
<td>2619</td>
<td>0%</td>
</tr>
<tr>
<td>mod/certificate/view.php</td>
<td>2364</td>
<td>0%</td>
</tr>
<tr>
<td>grade/report/user/index.php</td>
<td>2241</td>
<td>0%</td>
</tr>
<tr>
<td>mod/resource/view.php</td>
<td>2005</td>
<td>0%</td>
</tr>
<tr>
<td>mod/survey/view.php</td>
<td>1922</td>
<td>0%</td>
</tr>
<tr>
<td>mod/url/view.php</td>
<td>1887</td>
<td>0%</td>
</tr>
</tbody>
</table>
TURNING MOODLE INTO A PWA
CACHING PAGE VIEWS (II)

PROBLEM!
Many routes serve different content when authenticated / not-authenticated. Eg:
• While not authenticated all routes “serves” (redirect to) the login page.
• User-specific content (e.g.: same course route, different user/user role).

We could evaluate the session cookie credentials in our serving strategies...
But do we really want to cache the response (from an “authenticated” context) and make it publicly available to anyone with access to the browser?

¿App shell to the rescue?
TURNING MOODLE INTO A PWA CACHING PAGE VIEWS (III)

The App Shell approach (as I understand it) -> Refactor party!

• Remove all user-specific / authentication-needed data from ALL THE VIEWS AND LAYOUTS

• Render the view specific information using asynchronously called web services (with the proper login and capabilities checks)

• Render the layout “user-related” elements and information the same way (e.g.: the user menu, the course navigation panel...)
TURNING MOODLE INTO A PWA CACHING PAGE VIEWS (IV)

Refactoring party with the `mod_url/view.php` -> `/mod/url/view.php`
TURNING MOODLE INTO A PWA CACHING PAGE VIEWS (V)

+ web service to serve the mod_url view.php content (all “view” cases)

mod/url/classes/external.php
load_view
TURNING MOODLE INTO A PWA CACHING PAGE VIEWS (VI)

App-shelled mod_url view >>>

User, session... related information removed from the views
No user menu, no navigation menu, no footer user-related links, no mod_url view-specific data...

This is our “app shell”, let’s cache this! (stalewhilereval? cache first?)
TURNING MOODLE INTO A PWA CACHING PAGE VIEWS (VII)

Then we load asynchronously the actual content behind auth and capabilities checks.

This is the mod_url view content loaded from a web service.

Do the same with the information removed from the layout! -> web service + async loading.
TURNING MOODLE INTO A PWA
CACHING PAGE VIEWS (VIII)

Lighthouse audit (without refactoring the layout, just the mod_url view)

App shell approach:
“First meaningful paint”
~600 ms (half the time)

Caveats:
The actual content
“appears” much later
(+ 1 sec at least) (on load)
TURNING MOODLE INTO A PWA
ADDING THE SERVICE WORKER TO THE MOODLE CORE?

• Allow for plugins to define their scope and tie plugin routes to different caching strategies / precaching (ServiceWorker API ?).

• Create a php script that builds and serves a “revisioned” sw.js collecting and including all those plugin definitions across all the Moodle instance plugins.

• Add admin settings to easily enable/disable the inclusion of the service worker.

• Add admin settings to easily include JS code that unregisters previously added service workers and/or force clients to clear their storages and caches (in case something went wrong...).
Thanks to (Powered by)...

• 3iPunt: Pau Plana, Ebrahim Mesleh & Antoni Bertran

• 3iPunt “Moodle Team”:

  Eva Pereira, Raúl Martínez & Roser Pruaño

• All developers that have been documenting their experiences with Service Workers since 2015.