Charles Bochet Follow
@charlesBochet CTO @LuckeyHomes
Jun 30, 2017 · 8 min read

A Progressive Web Application with Vue JS, Webpack & Material Design [Part 3]

A Progressive Web App with VueJS, Webpack & Material Design
Part III
Offline experience with Service Workers

[Updated on 11/27/2017]

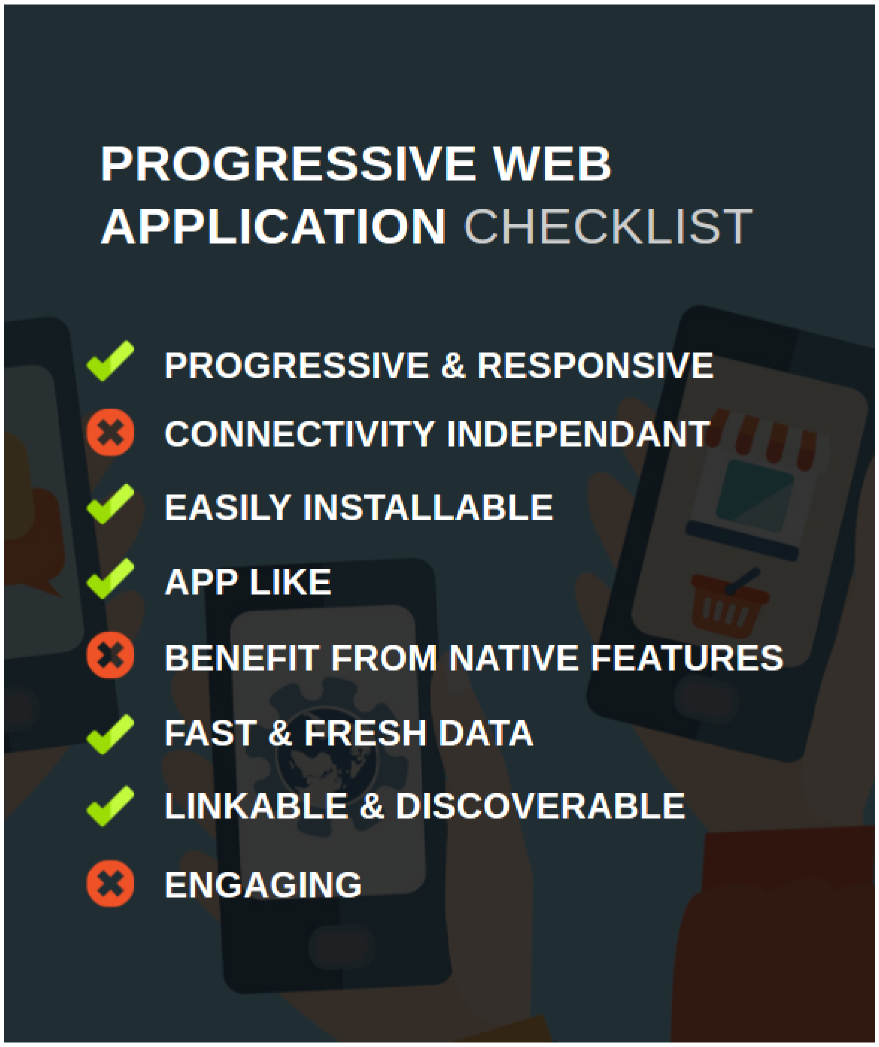
This article is part of a serie that aims to build a basic but complete Progressive Web Application with VueJs, Webpack & Material Design, step-by-step and from scratch. If you have not read them yet, you can find previous parts below:

- [Part 1] Create a Single Page Application with VueJS, Webpack and Material Design Lite
- [Part 2] Connect the App with a distant API with Vue-Resource and VueFire (by Kevin Jean)

Code source is available on this GitHub repository.

For those who have already followed first parts: I have updated Cropchat sources to use the brand new `vue init pwa` command. If you already cloned this repository, I strongly recommend that you pull this repository again.

Here is where we stopped:


PROGRESSIVE WEB APPLICATION CHECKLIST
✓ PROGRESSIVE & RESPONSIVE
✗ CONNECTIVITY INDEPENDANT
✓ EASILY INSTALLABLE
✓ APP LIKE
✗ BENEFIT FROM NATIVE FEATURES
✓ FAST & FRESH DATA
✓ LINKABLE & DISCOVERABLE
✗ ENGAGING

This third part aims to give our CropChat application a new dimension: **offline mode**.

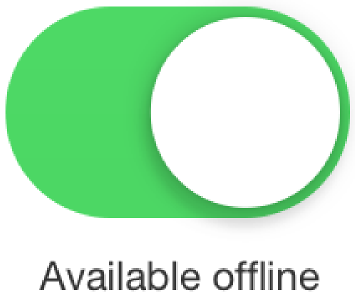
In a few words, we are going to learn:

- What a Service Worker is and how it can offer offline experience ;
- How to cache your application App Shell (core js and css) ;

- How to cache your HTTP requests (external assets, cat pictures, …) ;
- How to cache Firebase stream.

Enjoy reading!

[PART 3] Offline experience with Service Workers



Being independant of network connectivity is one of the PWA key features. We want to provide our users with an app-like user experience and keep them engaged: we don't want them to be disconnected or frozen each time they have slow or nonexistent connectivity.

Unable to connect to the Internet

Google Chrome can't display the webpage because your computer isn't connected to the Internet.

Chrome usual behavior when user tries to reach a web application with a low or nonexistent connectivty

A Progressive Web App is basically a web application that runs in a webview. However, it benefits from recent browsers & OS improvements that make offline mode possible: **Service Workers.**

Service Workers: a short introduction

A **Service Worker** is a JS script that user's browser runs in the background, separately from your web application. A Service Worker executes code even when your application is closed or inactive, and opens the way to app-like features such as **caching** (using Cache Storage API and Fetch API), **push notifications** (using Push API) or **background sync**.

If you want to learn more about Service Workers, please have a look at Google documentation.

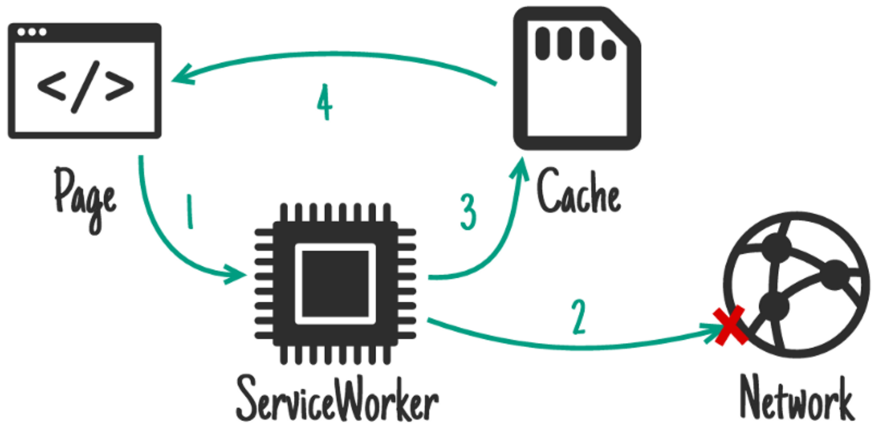
Remark: Progressive applications do not necesserly implement an offline mode on every device: iOS do not yet support Service Workers. This is the reason why we call them **progressive**: they will run on every device and propose a better/advanced user experience if the device can handle it.

How does it work?

You can see a Service Worker as a proxy. Each HTTP request that our application makes triggers a `fetch` event. **This event is catched by our Service Worker who choose to retrieve from cache or fetch from the network.**

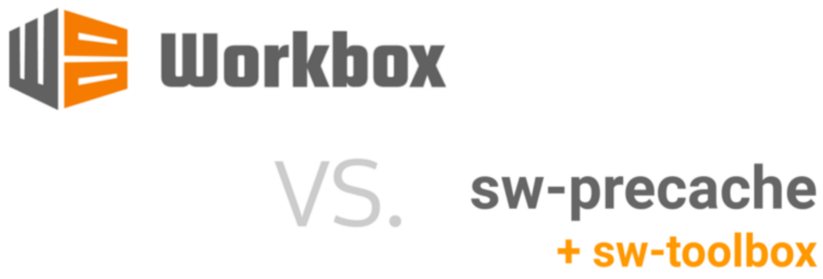
All you have to do is to populate your cache when your application is online. How does your Service Worker knows when to cache and when to fetch from network? It's up to you: you have precise control on **caching strategies**: cache first (useful to cache App shell files), network first (useful to cache remote assets or data)…

For example, here is a "classic" caching strategy called Network first:



There are many other caching strategies but I won't cover them here: for further information, refer to Google Offline Coobook.

Service Worker Libraries and VueJS



While writing this article, I tried several caching methods. Here are your choices:

- writing your Service Worker manually following Google code proof of concept ;

- use sw-precache library for static caching and sw-toolbox (that comes along with sw-precache) for runtime caching ;
- use the new workbox plugin developed by Google.

Here is my advice: **use sw-precache plugin**. Writing your own Service Worker is slow and repetitive: it is not a viable solution. Workbox plugin is promising but is not yet ready [June 2017]: it does not yet support runtime caching. [**UPDATE**: it seems that they released it 3 days ago. Have to look at it!]

Let's go!

Step 1: Cache your own static assets with sw-precache

`sw-precache` is able to cache static assets that are served by your own application : your **App Shell**. The App Shell is "the minimal HTML, CSS and JavaScript required to power the user interface" following Google's App Shell Model.

This way, no matter what network connectivity you have, your core assets will be loaded and your Front Application will be up and running. This is named **pre-caching.**

We are going to use sw-precache-webpack-plugin. To install it:

```
npm install sw-precache-webpack-plugin --save
```

Configure sw-precache

Actually, new VueJS cli template already includes `sw-precache` . If you are using it, you won't even need to install `sw-precache` .

Have a look to your `dist/webpack.prod.conf.js` file and update your production webpack configuration:

```
var SWPrecacheWebpackPlugin = require('sw-precache-webpack-

...

var webpackConfig = merge(baseWebpackConfig, {
  ...
  plugins: [
    ...
    // service worker caching
    new SWPrecacheWebpackPlugin({
      cacheId: 'my-vue-app',
      filename: 'service-worker.js',
```

Add following line to `<body>` section in your `index.html` so Webpack will load `sw-precache` plugin.

```
<%= htmlWebpackPlugin.options.serviceWorkerLoader %>
```

Build Cropchat in production mode:

```
npm run build
```

That's all you need to cache Cropchat App Shell.

Behind the scene

During CropChat's build, `sw-precach-webpack-plugin` generates a `service-worker.js` file for us. This Service Worker will be located in `dist/service-worker.js` and will be loaded and executed in your browser on first application run. Have a look at it:

```
"use strict";

var precacheConfig = [
  ["index.html", "218187414cc275eaa7bb37f098e0fc92"],
  ["static/css/app.d1a62a5e00430713ca6ccd67cb5fd580.css", "
  ["static/js/app.04ef0c51d385926ea560.js", "13cfbbcfd70773
  ["static/js/manifest.2d92f59992387f01763b.js", "00451931f
  ["static/js/vendor.0140506021c3a9c89dd2.js", "27f61c490b1
];
...
urlsToCacheKeys = new Map(precacheConfig.map(function (e) {
  var t = e[0], n = e[1], r = new URL(t, self.location), a
  return [r.toString(), a]
}));

self.addEventListener("install", function (e) {
  ...
});

self.addEventListener("activate", function (e) {
  ...
})

self.addEventListener("fetch", function (e) {
  if ("GET" === e.request.method) {
```

Test offline mode

You probably have noticed that I am not taking care of `webpack.dev.conf.js` file. There is a good reason why. Service Workers is a production feature: you don't want `*.js` and `*.css` files to be cached while developping your application.

To test it, build Cropchat in production mode:

```
npm run build
```

In order to emulate an HTTP server and serve Cropchat built files, you can use brilliant serve tool: (requires node ≥ 6.9.0)

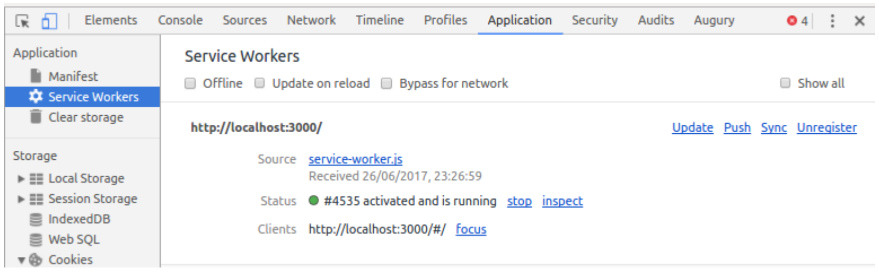
```
sudo npm install -g serve
```

Tthen, let the magic happen:

```
serve dist/
```

And browse: http://localhost:3000

Press F12 to open Chrome Developer Tools and open Application Tab:



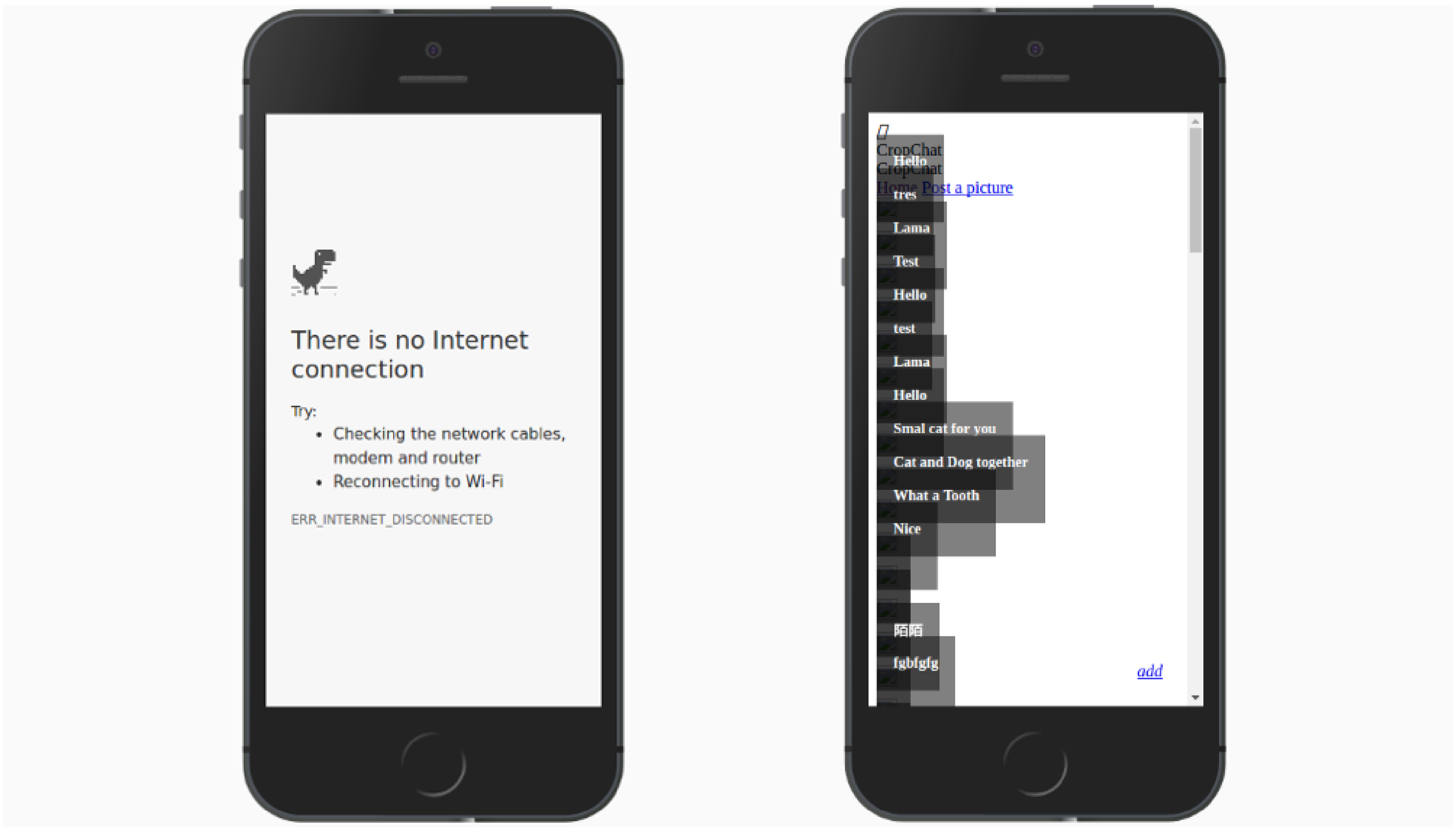
http://localhost:3000/
Source service-worker.js
Received 26/06/2017, 23:26:59
Status #4535 activated and is running
Clients http://localhost:3000/#/

Chrome Developer Tools: our Service Worker is installed

Under Service Workers menu, you will see that CropChat now has a Service Worker installed and running. Chrome Developer Tools is a very powerful debugging software. If you want to learn more about Service Workers debugging, do not hesitate to have a look at this Progressive Web App section.

Now, let’s test offline mode:

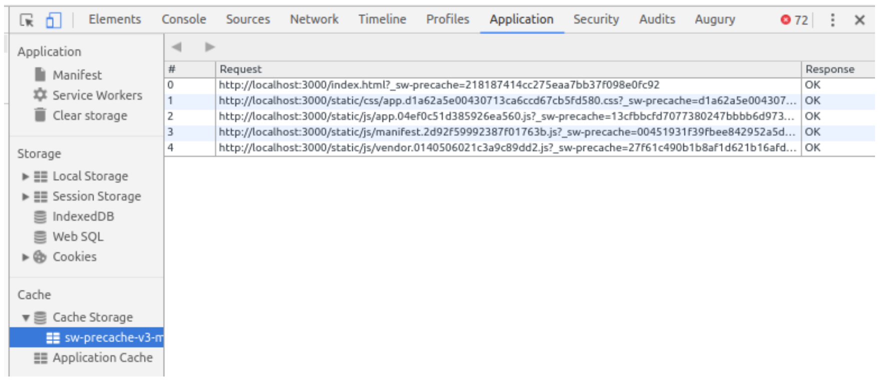
- Go back to **Service Workers** menu item, activate Offline Mode ;
- under **Network Tab**, see that Offline Mode is activated here too ;
- under **Network Tab**, disable cache (browser cache <> Service Worker cache) ;
- Refresh.



Without and with Service Workers

Application is still running thanks to our Service Worker that runs in background and serves assets rom cache.

Have a look to Cache Storage: (You may need to close and re-open dev tools):



#	Request	Response
0	http://localhost:3000/index.html?_sw-precache=218187414cc275eaa7bb37f098e0fc92	OK
1	http://localhost:3000/static/css/app.d1a62a5e00430713ca6ccd67cb5fd580.css?_sw-precache=d1a62a5e004307...	OK
2	http://localhost:3000/static/js/app.04ef0c51d385926ea560.js?_sw-precache=13cfbbcfd7077380247bbbb6d973...	OK
3	http://localhost:3000/static/js/manifest.2d92f59992387f01763b.js?_sw-precache=00451931f39fbee842952a5d...	OK
4	http://localhost:3000/static/js/vendor.0140506021c3a9c89dd2.js?_sw-precache=27f61c490b1b8af1d621b16afd...	OK

Chrome Developer Tools: Cache Storage

Static assets are cached!

However, we are far from a proper offline mode! Here is why:

- we are not caching external Material Design Lite fonts and stylesheets, fetched from https://code.getmdl.io/1.2.1/material.blue-red.min.css ;
- we are not caching cats pictures ;

We are going to address these points in the next section.

Step 2: Cache HTTP requests with sw-toolbox

Basically, we would like to cache all HTTP requests. Service Workers can do that for us too. It's named **runtime caching**.

Cache external Material Design Lite assets

Again, we won't write our Service Worker manually. There is great plugin for that called `sw-toolbox` and it is already included in `sw-precache-webpack-plugin` . Nothing more to install: isn't VueJS + webpack combo great?

Update your `build/webpack.prod.conf.js` :

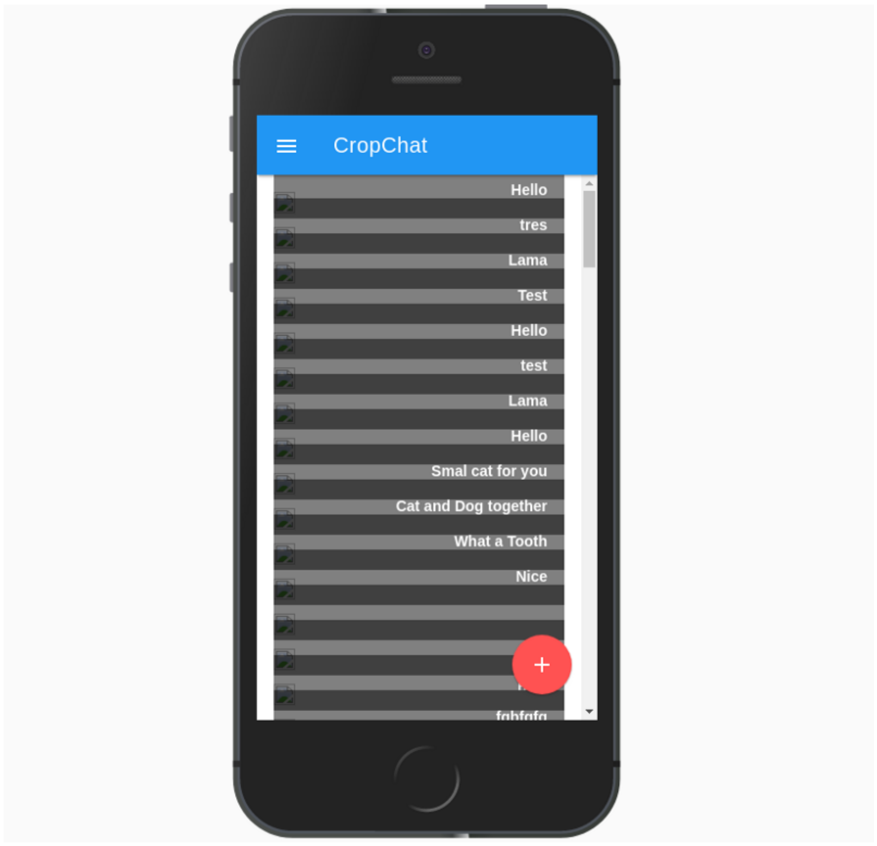
```
var SWPrecacheWebpackPlugin = require('sw-precache-webpack-

...

var webpackConfig = merge(baseWebpackConfig, {
  ...
  plugins: [
    ...
    // service worker caching
    new SWPrecacheWebpackPlugin({
      cacheId: 'my-vue-app',
      filename: 'service-worker.js',
      staticFileGlobs: ['dist/**/*.{js,html,css}'],
      minify: true,
      stripPrefix: 'dist/',
      runtimeCaching: [
      {
        urlPattern: /^https:\/\/fonts\.googleapis\.com\//,
        handler: 'cacheFirst'
      },
```

Runtime caching with sw-precache-webpack-plugin

Build and serve again.
You may need to Unregister your service worker, close your Chrome Browser… (Service Workers lifecycle seems a bit buggy right now)

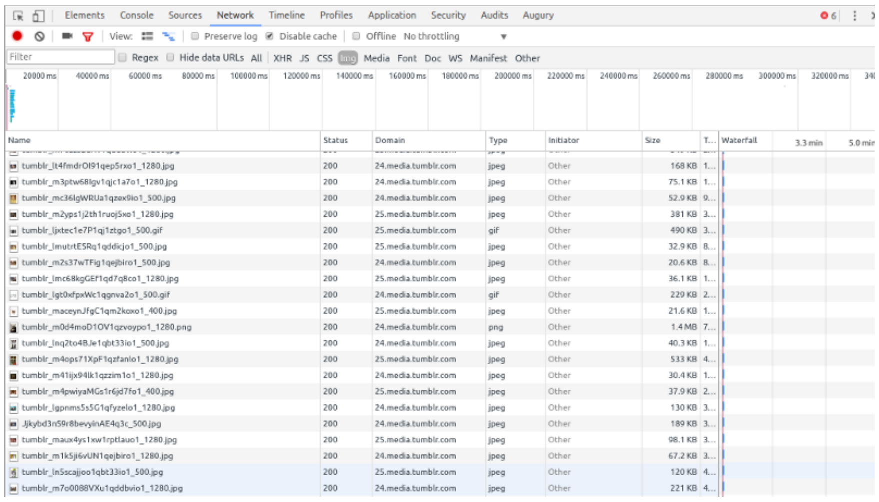


Hello
tres
Lama
Test
Hello
test
Lama
Hello
Smal cat for you
Cat and Dog together
What a Tooth
Nice
fghfgfg

Service Worker with runtime caching

Cache Cropchat cat images

We are close to achieving our goal. All we need to do is to cache CropChat cat images. Remember, cat pictures urls are fetched from the Cat API. Here is what they look like:



Preserve log: ☐ Disable cache: ✓ Offline: ☐ No throttling

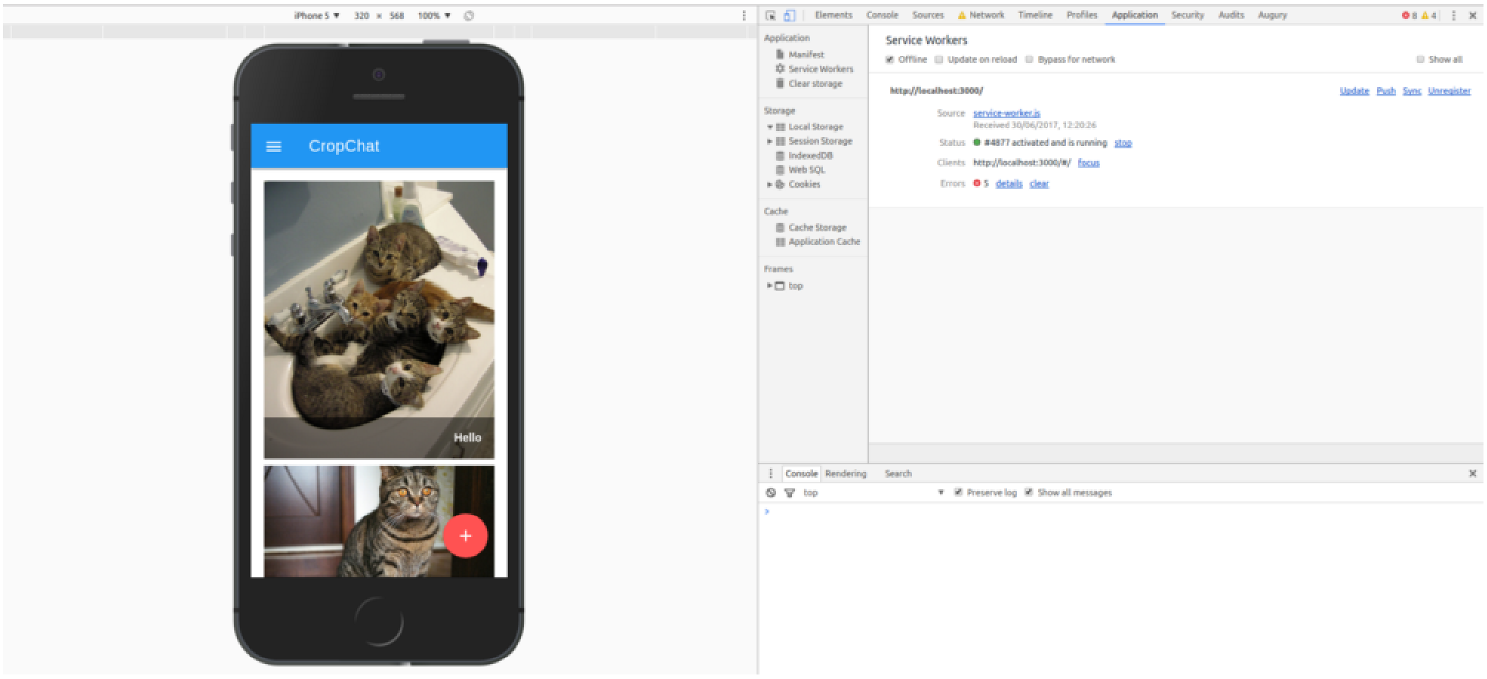
Name	Status	Domain	Type	Initiator	Size	T...	Waterfall
tumblr_lt4fmdrOI91qep5rxo1_1280.jpg	200	24.media.tumblr.com	jpeg	Other	168 KB	1...	
tumblr_m3ptw68lgv1qjc1a7o1_1280.jpg	200	24.media.tumblr.com	jpeg	Other	75.1 KB	1...	
tumblr_mc36lgWRUa1qzex9io1_500.jpg	200	24.media.tumblr.com	jpeg	Other	52.9 KB	9...	
tumblr_m2yps1j2th1ruoj5xo1_1280.jpg	200	25.media.tumblr.com	jpeg	Other	381 KB	3...	
tumblr_ljxtec1e7P1qj1ztgo1_500.gif	200	25.media.tumblr.com	gif	Other	490 KB	3...	
tumblr_lmutrtESRq1qddicjo1_500.jpg	200	25.media.tumblr.com	jpeg	Other	32.9 KB	8...	
tumblr_m2s37wTFig1qejbiro1_500.jpg	200	24.media.tumblr.com	jpeg	Other	20.6 KB	8...	
tumblr_lmc68kgGEf1qd7q8co1_1280.jpg	200	25.media.tumblr.com	jpeg	Other	36.1 KB	1...	
tumblr_lgt0xfpxWc1qgnva2o1_500.gif	200	24.media.tumblr.com	gif	Other	229 KB	2...	
tumblr_maceynJfgC1qm2koxo1_400.jpg	200	25.media.tumblr.com	jpeg	Other	21.6 KB	1...	
tumblr_m0d4moD1OV1qzvoypo1_1280.png	200	24.media.tumblr.com	png	Other	1.4 MB	7...	
tumblr_lnq2to4BJe1qbt33io1_500.jpg	200	24.media.tumblr.com	jpeg	Other	40.3 KB	1...	
tumblr_m4ops71XpF1qzfanlo1_1280.jpg	200	25.media.tumblr.com	jpeg	Other	533 KB	4...	
tumblr_m41ijx94lk1qzzim1o1_1280.jpg	200	24.media.tumblr.com	jpeg	Other	30.4 KB	1...	
tumblr_m4pwiyaMGs1r6jd7fo1_400.jpg	200	25.media.tumblr.com	jpeg	Other	37.9 KB	2...	
tumblr_lgpnms5s5G1qfyzelo1_1280.jpg	200	24.media.tumblr.com	jpeg	Other	130 KB	3...	
Jjkybd3n59r8bevyinAE4q3c_500.jpg	200	24.media.tumblr.com	jpeg	Other	189 KB	3...	
tumblr_maux4ys1xw1rptlauo1_1280.jpg	200	25.media.tumblr.com	jpeg	Other	98.1 KB	3...	
tumblr_m1k5ji6vUN1qejbiro1_1280.jpg	200	24.media.tumblr.com	jpeg	Other	67.2 KB	3...	
tumblr_ln5scajjoo1qbt33io1_500.jpg	200	25.media.tumblr.com	jpeg	Other	120 KB	4...	
tumblr_m7o0088VXu1qddbvio1_1280.jpg	200	24.media.tumblr.com	jpeg	Other	221 KB	4...	

Chrome Developer Tools: Network tab gives us CropChat images urls list

Update `webpack.prod.conf.js:`

```
runtimeCaching: [
    {
        urlPattern: /^https:\/\/thecatapi\.com\/api\/images\/
        handler: 'cacheFirst'
    },
    {
        urlPattern: /^https:\/\/(\d+)\.media\.tumblr\.com\//,
        handler: 'cacheFirst'
    },
    {
```

Re-build, serve your application again (unregister your Service Worker or/and close Chrome if needed), activate offline mode and test it again:



Service Workers

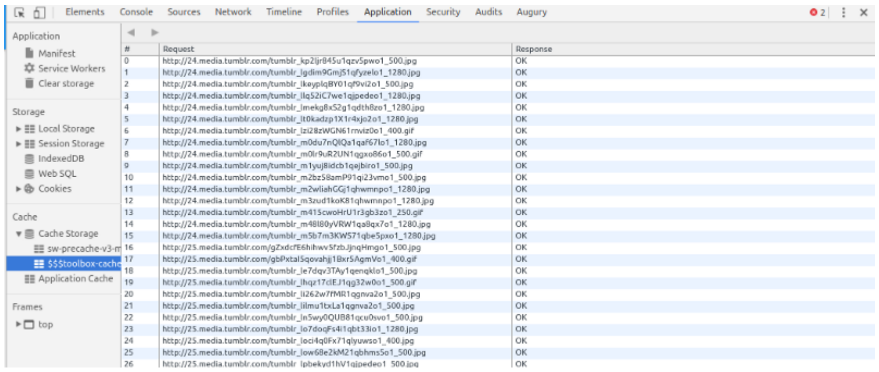
[✓ Offline] Update on reload Bypass for network Show all

http://localhost:3000/ Update Push Sync Unregister

Source service-worker.js
Received 30/06/2017, 12:20:26
Status ● #4877 activated and is running stop
Clients http://localhost:3000/#/ focus
Errors ● 5 details clear

CropChat is now available offline!

Take a look at your browser's Cache Storage again:



#	Request	Response
0	http://24.media.tumblr.com/tumblr_kp2ljr845u1qzv5pwo1_500.jpg	OK
1	http://24.media.tumblr.com/tumblr_lgdim9GmjS1qfyzelo1_1280.jpg	OK
2	http://24.media.tumblr.com/tumblr_lkeyplqBY01qf9vi2o1_500.jpg	OK
3	http://24.media.tumblr.com/tumblr_llq52iC7we1qjpedeo1_1280.jpg	OK
4	http://24.media.tumblr.com/tumblr_lmekg8xS2g1qdth8zo1_1280.jpg	OK
5	http://24.media.tumblr.com/tumblr_lt0kadzp1X1r4xjo2o1_1280.jpg	OK
6	http://24.media.tumblr.com/tumblr_lzi28zWGN61rnviz0o1_400.gif	OK
7	http://24.media.tumblr.com/tumblr_m0du7nQlQa1qaf67lo1_1280.jpg	OK
8	http://24.media.tumblr.com/tumblr_m0lr9uR2UN1qgxo86o1_500.gif	OK
9	http://24.media.tumblr.com/tumblr_m1yuj8idcb1qejbiro1_500.jpg	OK
10	http://24.media.tumblr.com/tumblr_m2bz58amP91qi23vmo1_500.jpg	OK
11	http://24.media.tumblr.com/tumblr_m2wliahGGj1qhwmnpo1_1280.jpg	OK
12	http://24.media.tumblr.com/tumblr_m3zud1koK81qhwmnpo1_1280.jpg	OK
13	http://24.media.tumblr.com/tumblr_m415cwoHrU1r3gb3zo1_250.gif	OK
14	http://24.media.tumblr.com/tumblr_m48l80yVRW1qa8qx7o1_1280.jpg	OK
15	http://24.media.tumblr.com/tumblr_m5b7m3KW571qbe5pxo1_1280.jpg	OK
16	http://25.media.tumblr.com/gZxdcfE6hihwv5fzbJjnqHmgo1_500.jpg	OK
17	http://25.media.tumblr.com/gbPxtal5qovahjj1Bxr5AgmVo1_400.gif	OK
18	http://25.media.tumblr.com/tumblr_le7dqv3TAy1qenqklo1_500.jpg	OK
19	http://25.media.tumblr.com/tumblr_lhqz17clEJ1qg32w0o1_500.gif	OK
20	http://25.media.tumblr.com/tumblr_li262w7fMR1qgnva2o1_500.jpg	OK
21	http://25.media.tumblr.com/tumblr_lilmu1txLa1qgnva2o1_500.jpg	OK
22	http://25.media.tumblr.com/tumblr_ln5wy0QUB81qcu0svo1_500.jpg	OK
23	http://25.media.tumblr.com/tumblr_lo7doqFs4i1qbt33io1_1280.jpg	OK
24	http://25.media.tumblr.com/tumblr_loci4q0Fx71qlyuwso1_400.jpg	OK
25	http://25.media.tumblr.com/tumblr_low68e2kM21qbhms5o1_500.jpg	OK
26	http://25.media.tumblr.com/tumblr_lpbekyd1hV1qjpedeo1_500.jpg	OK

Chrome Developer Tools: CropChat Cat Images are now cached.

That's it!

Step 3: Cache Firebase WebSocket with Local Storage API

I lied to you. Our application is not yet functionnal online. Try to turn on your network connection (instead of using Chrome Offline Mode): you won't see CropChat content anymore.

Indeed, Firebase stream is not cached. Firebase protocole is based on Web Sockets and **Service Workers can't handle WebSockets**.

We need to handle Firebase case manually. Here is my network-first caching strategy:

- if user has access to the Internet (**using navigator.onLine check**): cache Firebase Websocket content in browser's local storage, then hook on Firebase WebSocket ;
- else: serve cached content from browser's local storage.

To get it done, update your `HomeView.vue` component with the following code:

```
<script>
  export default {
    methods: {
      displayDetails (id) {
        this.$router.push({name: 'detail', params: { id: id
      },
      getCats () {
        if (navigator.onLine) {
          this.saveCatsToCache()
          return this.$root.cat
        } else {
          return JSON.parse(localStorage.getItem('cats'))
        }
      },
      saveCatsToCache () {
        this.$root.$firebaseRefs.cat.orderByChild('created_
          let cachedCats = []
          snapchot.forEach((catSnapchot) => {
            let cachedCat = catSnapchot.val()
            cachedCat['.key'] = catSnapchot.key
```

And update <template> section to use `getCats()` in `v-for` loop.

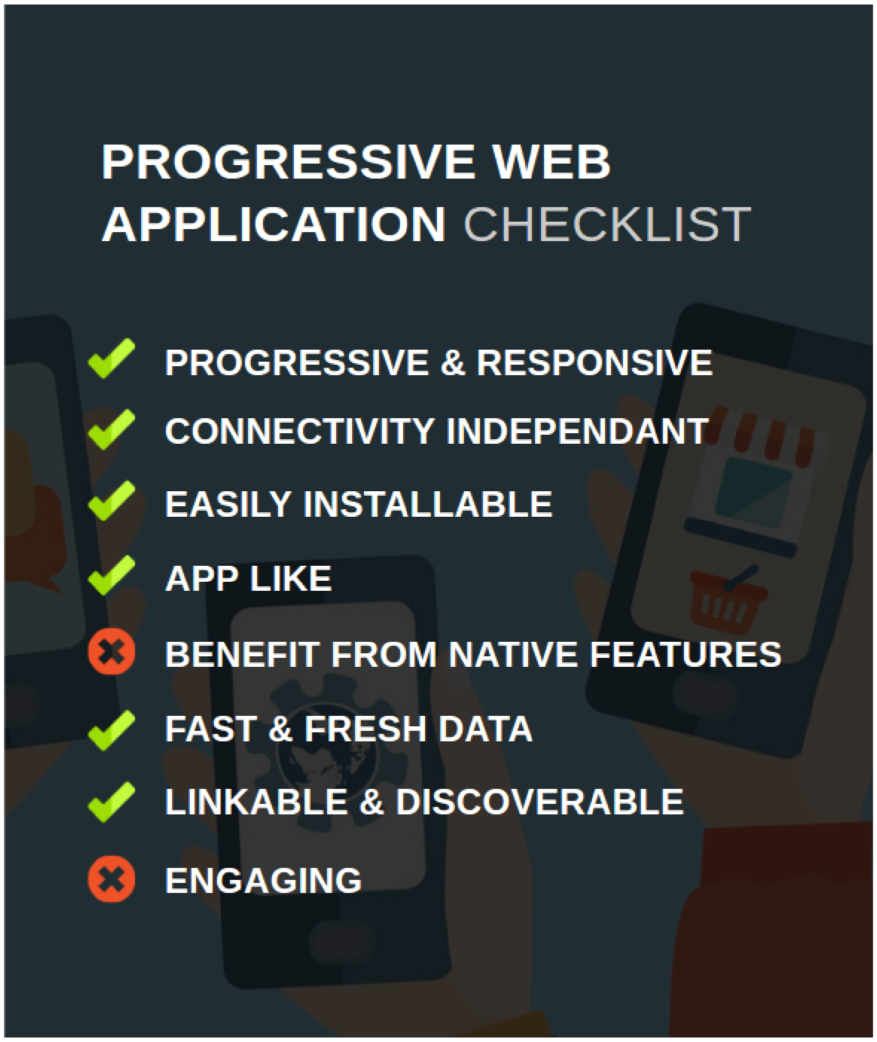
Build your app again, serve it, load it (with internet connection back) and try it offline !

Conclusion

I hope you have a better understanding of how to implement offline mode with VueJS, Webpack and a few plugins. We learned:

- how to use `sw-precache` to cache App Shell files ;
- how to user `sw-toolbox` through `runtimeCaching` to cache external assets ;
- how to handle Firebase websockets caching using browser's local storage.

Let's have a look to our Progressive Web App checklist:


PROGRESSIVE WEB APPLICATION CHECKLIST
✓ PROGRESSIVE & RESPONSIVE
✓ CONNECTIVITY INDEPENDANT
✓ EASILY INSTALLABLE
✓ APP LIKE
✗ BENEFIT FROM NATIVE FEATURES
✓ FAST & FRESH DATA
✓ LINKABLE & DISCOVERABLE
✗ ENGAGING

Two of our requirements are not yet met. We will handle them in next parts. Part 4 will show you how to "benefit from native features" using HTML 5 powerful APIs such as MediaDevice and MediaStream APIs.

For those who are in Paris, I am co-organizing a Progressive Web App MeetUp. Do not hesitate to say 'Hi!'.

This embedded content is from a site that does not comply with the Do Not Track (DNT) setting now enabled on your browser.

Please note, if you click through and view it anyway, you may be tracked by the website hosting the embed.

Did you like this article? Feel free to comment or contact me.