Progressive Enhancements

Let’s keep on talking about progressive enhancements in relation to our app that we are building. Notice that there’s a button for each article with a ‘read later’ label. Imagine a situation where the user is on a train, commuting to work, but there’s a part of the journey when there’s no network connectivity - maybe the train goes through a tunnel. In this case, it’d be great if we could save articles for offline reading. What’s great about our application structure so far is that we can add the progressive enhancements to `app.js` in a very easy fashion because all other logic has been moved out from this file.

"Read Later"

Let's implement the 'read later' functionality for our application.

From a technical perspective, allowing articles to be read later means that we want to add the article’s content to the cache proactively - this will give users a nice experience for our site - they get to choose which articles they wish to read when offline.

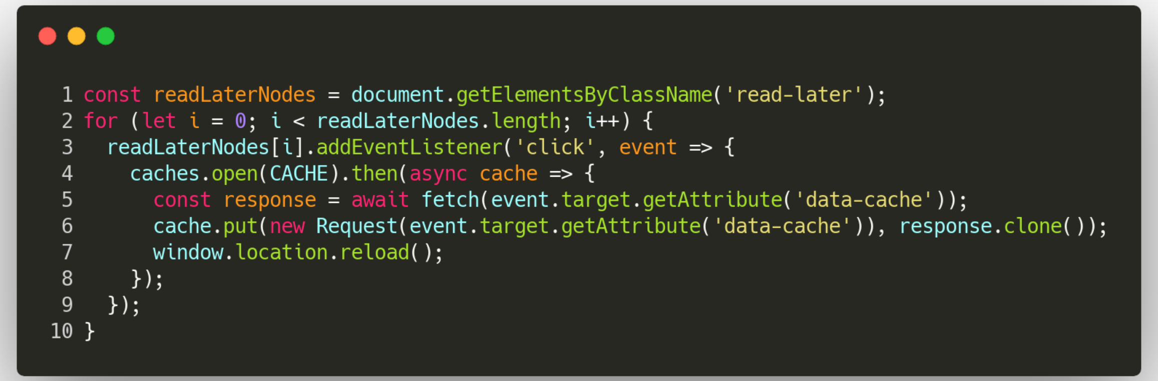
The question is, of course, how would this work? Well, it turns out that the cache interface is not only available in the context of the service worker, but it's also available in any other JavaScript programme, which means that we can programmatically add items to it, just like how we have done it from the service worker itself. And the beauty of this? The service worker doesn't care where the data comes from in the cache - it only cares about the fact whether it's there or not.

Let's take a look at how to implement this. For each button we’d need to add two things: a new attribute, identifying the article - which we can easily do by specifying the ID of the article, and we also need a click handler for the “read later” button itself.

The `data-cache` attribute will have a value that’s the full REST API URL for the article itself.

Note that we could have used any other `data-` attribute - I like `data-cache` because it indicates that we want to store data in the cache.

Since the read later buttons have a class reflecting their functionality, we can very easily add an event listener for each of the buttons by executing a loop. In that loop, we can open the same cache that the service worker uses to store API data. Once it's open, we can make a call to the RESTful API, and put the request along with the response to the cache. One thing to note here is that it's not enough to take the string value from the `data-cache` attribute, we need also to wrap it in a `new Request()` call because that should be the key that we store in the cache. Failing to do this would yield incorrect behaviour as well as unmatched values when doing a lookup based on the request. Why is this? Think about it, when the browser is making an actual request it does it in the form of `new Request()` and if we do a lookup in the cache a 'string of the request' is not going to be the same as `new Request()`.



```
const readLaterNodes = document.getElementsByClassName('read-later');
for (let i = 0; i < readLaterNodes.length; i++) {
  readLaterNodes[i].addEventListener('click', event => {
    caches.open(CACHE).then(async cache => {
      const response = await fetch(event.target.getAttribute('data-cache'));
      cache.put(new Request(event.target.getAttribute('data-cache')), response.clone());
      window.location.reload();
    });
  });
}
```

Finally also notice a little trick - `window.location.reload()`. Why is there a need to reload the window? It'll become clear after the next section. For now, the read later functionality is in place - feel free to test it.

Visual Cues

We are at a rather good state when it comes to our application; however, we can do a lot better. Let's keep on this track of "progressive enhancements". It would be nice to have some visual enhancements, indicating to users which articles can be read offline. This is going to be a small change but from a UX perspective a rather important one.

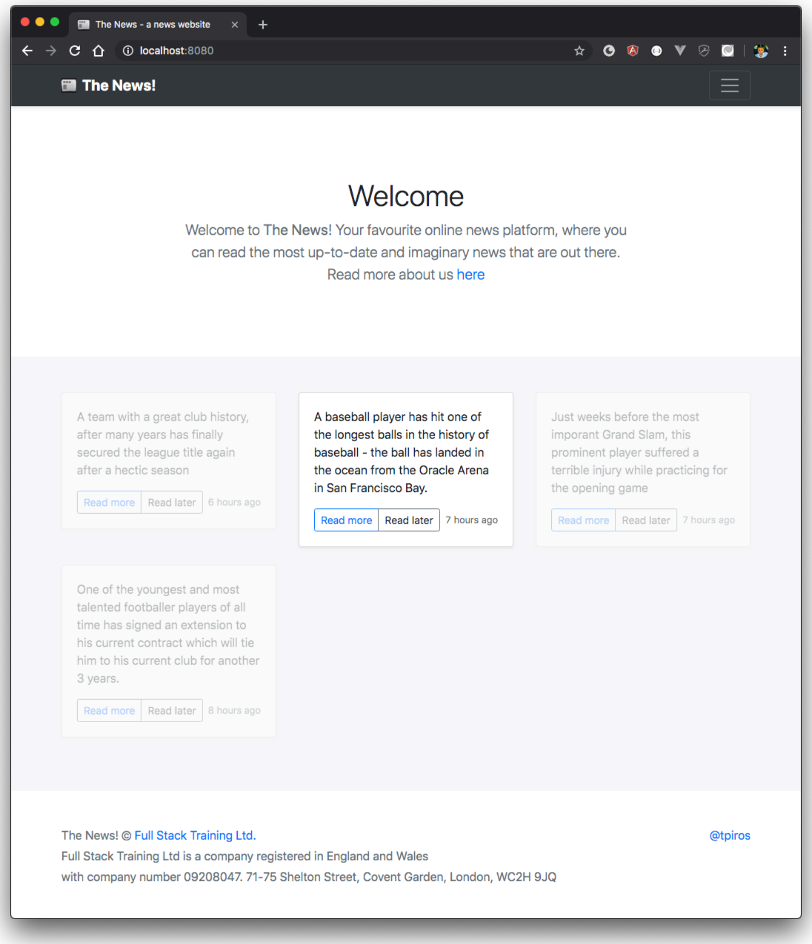
From a user experience perspective, at the moment there's no way to tell which article can be read offline so we could end up in a situation where a user may click on the read more button and not get any data. It'd be great if we'd provide our users with a visual cue indicating which articles they can access when offline - this includes articles that were visited already as well as articles that were chosen by them to be read later.

To implement the desired change, we can rely on the browser's offline/online state, which can easily be accessed via `navigator.onLine` and the two states that it can have, of course, are 'online' and 'offline. Furthermore, we can add an event listener to the window object to act when the online/offline state changes, and pass in a callback. The callbacks can check if any of the articles are in the cache and add opacity to the ones that are not present - almost hiding them from the user.

We can see that there's a missing link here - and that is - how do we know what's in the cache and what's not? We can add some helpers to our article cards via a custom data attribute - let's call this `data-cache`. This is going to work in the same way as we saw earlier with the read later button.

For adding/removing the opacity all we need to do is to open up the cache that stores our API data, iterate through the items in it and match those with the `data-cache` attribute values added to the cards.

Now, try to comment out the `window.location.reload()` line out and see how the app behaves. Essentially, if we don't force a reload, and we bring the application offline, the visual cues are not going to be active. That's because our logic lives in JavaScript and it already has been executed. We could place the check for the uncached cards inside the online/offline event listeners - that could be a viable alternative as well.



Welcome

Welcome to The News! Your favourite online news platform, where you can read the most up-to-date and imaginary news that are out there.
Read more about us here

A team with a great club history, after many years has finally secured the league title again after a hectic season
Read more | Read later | 6 hours ago

A baseball player has hit one of the longest balls in the history of baseball - the ball has landed in the ocean from the Oracle Arena in San Francisco Bay.
Read more | Read later | 7 hours ago

Just weeks before the most imporant Grand Slam, this prominent player suffered a terrible injury while practicing for the opening game
Read more | Read later | 7 hours ago

One of the youngest and most talented footballer players of all time has signed an extension to his current contract which will tie him to his current club for another 3 years.
Read more | Read later | 8 hours ago

The News! © Full Stack Training Ltd.
Full Stack Training Ltd is a company registered in England and Wales
with company number 09208047. 71-75 Shelton Street, Covent Garden, London, WC2H 9JQ
@tpiros

At this point, we have a great Progressive Web App. However, PWAs have a lot more that we can discuss. In fact, the next thing is going to be another app-like feature, installation.