# 4CS017 – Internet Software Architecture tutorial

## Browser caching

*What will you learn today?*

You will learn to cache/retrieve your weather data to/from your web browser.

## Part 1 – caching your weather data to the browser

There are several browser APIs to store data (Local Storage, Session Storage, IndexedDB or even good old fashion cookies). Today we will use the NoSQL store “Local Storage” to implement a “Cache then Network” strategy.

Here is my working example, for reference (you can look at the code etc.): <https://mi-linux.wlv.ac.uk/~in9352/weather/task3-client.html>

1. Your “Cache then network” strategy is as follow:

* If we have data in our browser,and data is less than x seconds old
  + Display browser data
* Else
  + Get fresh data from network (e.g. your PHP API)
  + Display fresh data
  + Store fresh data to browser cache

The above can be implemented by adding the following code to your **existing** HTML file. I have used colours to show what does what, as per algorithm above.

// Check browser cache first, use if there and less than 10 seconds old

if(localStorage.when != null

&& parseInt(localStorage.when) + 10000 > Date.now()) {

let freshness = Math.round((Date.now() - localStorage.when)/1000) + " second(s)";

document.getElementById("myWeather").innerHTML = localStorage.myWeather;

document.getElementById("myTemperature").innerHTML = localStorage.myTemperature;

document.getElementById("myLastUpdated").innerHTML = freshness;

// No local cache, access network

} else {

// Fetch weather data from API for given city

fetch('https://mi-linux.wlv.ac.uk/~in9352/weather/my-api.php?city=Wolverhampton')

// Convert response string to json object

.then(response => response.json())

.then(response => {

// Copy one element of response to our HTML paragraph

document.getElementById("myWeather").innerHTML = response.weather\_description;

document.getElementById("myTemperature").innerHTML = response.weather\_temperature;

document.getElementById("myLastUpdated").innerHTML = response.weather\_when;

// Save new data to browser, with new timestamp

localStorage.myWeather = response.weather\_description;

localStorage.myTemperature = response.weather\_temperature + '°';

localStorage.when = Date.now(); // milliseconds since January 1 1970

})

.catch(err => {

// Display errors in console

console.log(err);

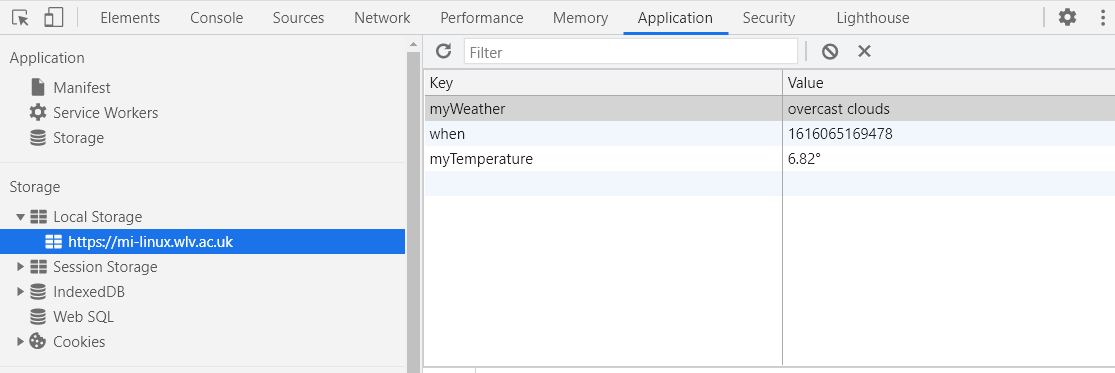
});

}

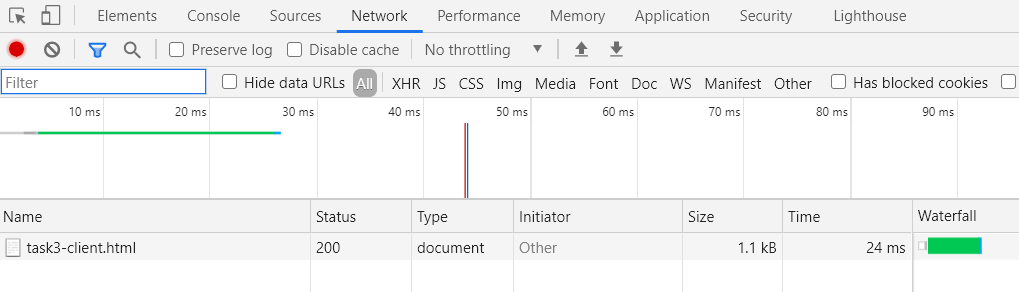
Things to note:

* + The if statement checks that the data in the browser cache is no older than x **milliseconds**. So, use 10 000 for 10 seconds, 60 000 for 1 minute etc.
  + You will have to tailor the different elements and fields (myWeather, myTemperature etc.) to **your** requirements, obviously (e.g. you might be displaying the pressure, not the temperature).
  + Make sure you “fetch” from **your own API** (as per prototype 2), this is the URL to mine.

1. Test that it works. After refreshing the page, you should see your weather data in the browser cache, under “Application / Local Storage / mi-linux”, like this:



Under the Network tab, you will notice that the second HTTP request (the one fetching data from your API) only happens every 10 seconds (or whatever time you’ve specified).



As such you should see a **significant performance improvement** compared to prototype 2, as for most requests we are not accessing PHP and MySQL at all.

1. Update your **UML Deployment Diagram**, by adding your service worker / cache artefacts in your existing “web browser” environment (on your local PC hardware node).

## Part 2 – Going further (important: for fun - **not** required for the assessment)

*“I have finished all the work above, what shall I do next?”*

In part 1 we amended our app so that it gets the **data** from the browser if available, but it still requires a network connection for the **files** (HTML, images etc.) Let’s go fully offline, using a **Service Worker**! My working example is located here: <https://mi-linux.wlv.ac.uk/~in9352/weather/task3-further.html>

Please note that the example below is based on the official Google doc – please read it for further details and explanations: <https://developers.google.com/web/fundamentals/primers/service-workers>

1. First, create a **new file** for your Service Worker (maybe “sw.js”), with the following code:

var CACHE\_NAME = 'my-site-cache-v1';

var urlsToCache = [

'task3-further.html'

];

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

// Perform install steps

event.waitUntil(

caches.open(CACHE\_NAME)

.then(function(cache) {

console.log('Opened cache');

return cache.addAll(urlsToCache);

})

);

});

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

event.respondWith(

caches.match(event.request)

.then(function(response) {

// Cache hit - return response

if (response) {

return response;

}

return fetch(event.request);

}

)

);

});

Things to note:

* The blue block of code should list **all** the files **your** app needs to function offline, including your HTML file, but also any images you might be using, external JS and CSS files etc. This is the only bit of code you need to change in the whole service worker.
* The orange block simply **installs** your service worker and **caches** all the files you have specified in the list above (see cache.addAll() method).
* The green block **intercepts** all network “fetches”, and checks if the requested file is in the cache. If it is, it fetches it from the cache. If not, it lets the request through to the network.
* Important: Service workers only work over **https**.

1. Next, you need to “include” your service worker in your main HTML file, by adding the following block at the beginning of your **existing** <script> section:

// Register service worker

if ('serviceWorker' in navigator) {

window.addEventListener('load', function() {

navigator.serviceWorker.register('sw.js').then(function(registration) {

// Registration was successful

console.log('ServiceWorker registration successful');

}, function(err) {

// registration failed :(

console.log('ServiceWorker registration failed: ', err);

});

});

}

The interesting bit is in blue… everything else is just error reporting etc.

1. Finally, if we fail to “fetch” data from the network, let’s always try and fetch it from the browser cache (if available), by adding this block of code in the existing error section.

So, this **existing** block of code:

.catch(err => {

// Display errors in console

console.log(err);

});

Becomes:

.catch(err => {

if(localStorage.when != null) {

// Get data from browser cache

let freshness = Math.round((Date.now() - localStorage.when)/1000) + " second(s)";

document.getElementById("myParagraph").innerHTML = localStorage.myParagraph;

document.getElementById("myTemperature").innerHTML = localStorage.myTemperature;

document.getElementById("myLastUpdated").innerHTML = freshness;

} else {

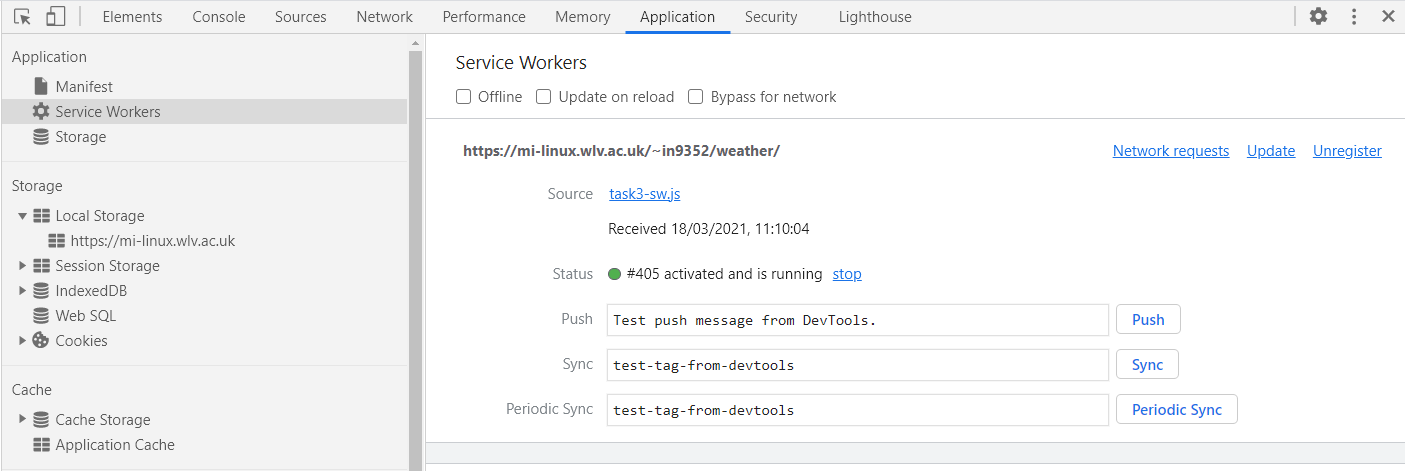
// Display errors in console

console.log(err);

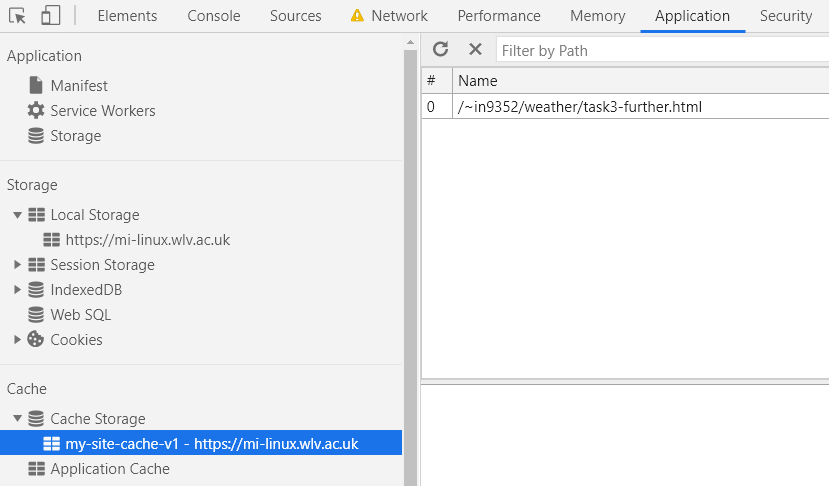
}

});

1. All done, let’s test! If you refresh your page, you should see your service worker “active and running” under “Application / Service Workers”:

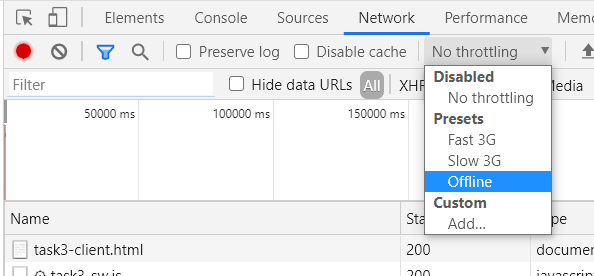


You can see the files cached by your service worker here:



Go offline:

* Go to the “Network” tab.
* Click on the “Throttling” dropdown and choose “Offline”:



* **Refresh** your page… it should **still work** and be **really fast**, as it’s getting both the files AND the data from the browser cache!
* **Important**: your service worker will **always** get the files from the cache, possibly causing some confusion: you might make a change to your HTML file, but it won’t show… you will need to “Unregister” your Service Worker every time you make a change!
* **Further work**:
  + We are not handling being offline really well, as we allow our data “fetch” to fail (see error in console). It doesn’t really matter, the user won’t notice, but you could improve your algorithm from part 1 to always get data from the browser cache when offline.
  + If offline for a long time, your data freshness will still show in seconds, leading to silly messages (e.g. data is 5671 seconds old). Use basic math and logic to convert the number to hours, minutes and seconds.