Week 4: IndexedDB Lab class
Introduction to the Code

Prof. Fabio Ciravegna
Department of Computer Science
University of Sheffield
f.ciravegna@shef.ac.uk

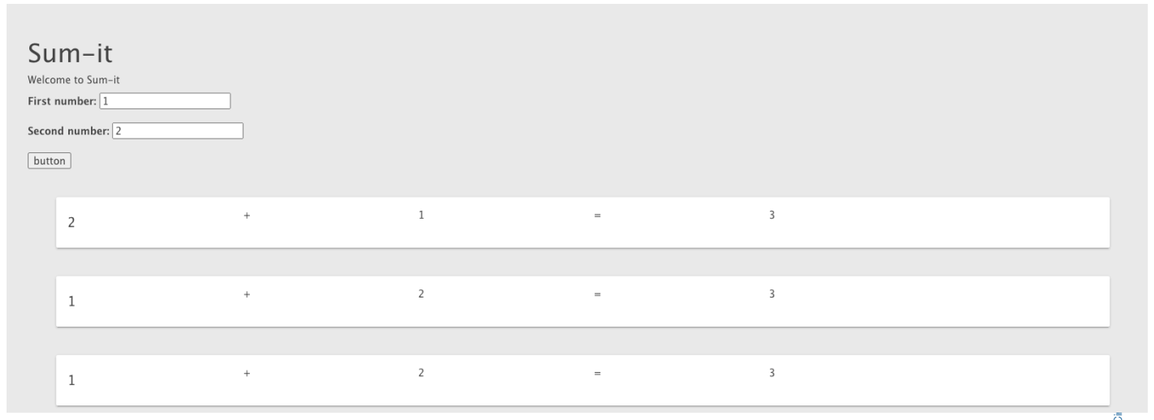
Aims of the exercise

- The aim of the class is to learn to use IndexedDB to store data on the client
- There are two parts in this exercise
  - Understanding how IndexedDB works
    - project: *Week 4.a Lab Class IndexDB* - ***Introduction***
      - *this is a very simple complete project* to inspect
    - project: Week 4.b - IndexedDB - **Exercise**
      - this is the actual exercise

- Introduction

- You are given a client server architecture that servers a simple EJS file containing a form
  - the form allows to input two numbers and generates the sum
  - then it stores the numbers and the sum into an IndexedDB
  - when a sum is inputted all the sums with the same results that were already stored in the database are displayed
- For example:
  - input: 1+2 = 3 => store {no1: 1, no2: 2, sum: 3} into the DB
  - input: 2+1 = 3 =>
    - store {no1: 2, no2: 1, sum: 3} into the DB
    - display {no1: 1, no2: 2, sum: 3} and {no1: 2, no2: 1, sum: 3}
      - note as the operation is asynchronous it is possible that the current input is not displayed and only the stored values are displayed

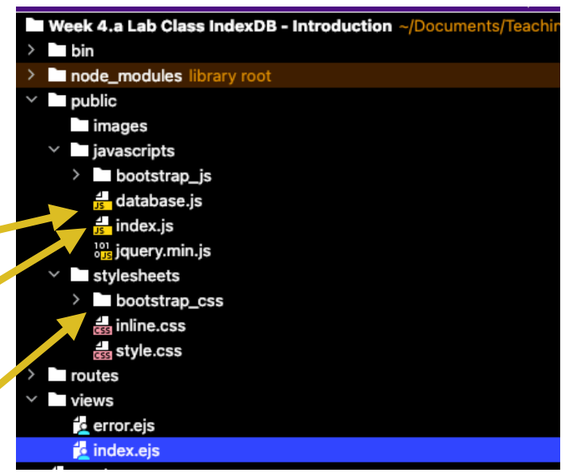
Example



The code

- the form return false so not to post to the server
- the form button calls a function *submitForm()*
  - the function serialises the form to create a data structure like:
    - {no1: 2, no2: 1, sum: 3}
  - the data is stored into the IndexedDB
    - storeSumData({no1: data.no1, no2: data.no2, sum: sum});
  - then it retrieves the past operations with the same result
    - getSumData(sum);

The project structure



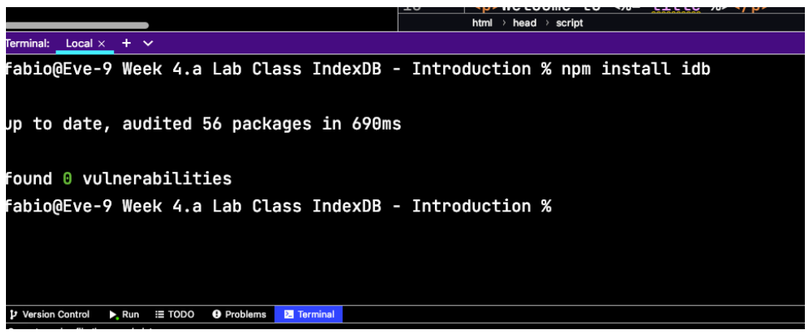
the functions accessing the database

the main JS functions

cached bootstrap library
(results presentation)

Import the idb module

- open the terminal in WebStorm
- type **npm install idb**

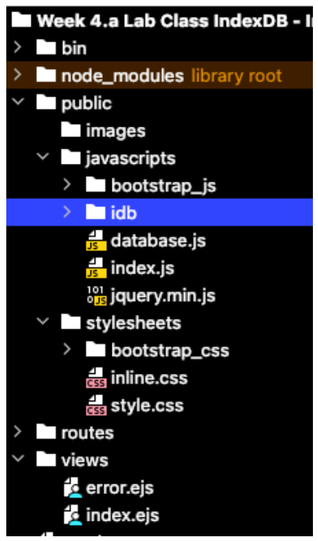


We cache all libraries

- it will be useful in the next exercises when we will work offline with service workers
  - so try and keep everything local
  - it has also the non minor advantage that if the library changes, your programme is not at risk of breaking

Copy the idb/build folder locally

- As we will work offline, we need to store everything locally:
- For idb do the following:
  - open *node_modules/idb*
  - copy the folder *build* into the folder *public/javascripts*
  - rename it as *idb*
- Note: this has been already done for you in the current project
  - but you should do it every time you need to work offline with idb in a new project
    - e.g. in your assignment



Modules in Javascript

As many of you may have never worked with a module before especially browser side

What is a module?

- JavaScript programs started off pretty small
  - mostly used initially to do isolated scripting tasks,
  - providing interactivity to your web pages
  - large scripts were generally not needed
- Today complete applications are run also outside the browser, e.g. Node.js
- So it makes sense to create modules
  - Providing mechanisms for splitting JavaScript programs up into separate modules that can be imported when needed
  - Good modules are highly self-contained with distinct functionality, allowing them to be shuffled, removed, or added as necessary, without disrupting the system as a whole

Modules = Name spaces

- Modules are similar in concept toJava packages
  - rather than classes
  - they define name spaces that expose public methods and variables
    - external namespaces cannot see other than public items
  - Exporting functions and variables is a mechanism similar to the public methods in Java
    - only methods that are public are visible outside a package

How to use modules in Node.JS

- We have seen this already
  - first you *npm install module_name*
  - the use it in the code using *require*

```
var express = require('express');
```

  - *then you can use its **exported** functions and variables*
    - *express.functionName*
- To export in node.js:

```
// in module: Log.js
 module.exports = function (msg) {
    console.log(msg);
};
------
//in another nodejs  file
var msg = require('./Log.js');
msg('Hello World');
```

In this case the export is unnamed
you can also name it. We will see examples in the MongoDB lecture

Browser side

- Importing in Javascript

```
import * as idb from './idb/index.js';
```

  - * imports all the exported functions and variables
    - they can be retrieved using the namespace idb
      - example:

```
idb.openDB(SUMS_DB_NAME, 2,   …
```

  - note: the import function can only be used in another JS module

Browser side

- Importing a module from HTML or EJS

```
<script src="/javascripts/database.js" type="module"></script>
```

  - note type=module
- If your JS file imports from a module it must be a module itself
  - You must always import it as type module in your HTML (or EJS) files
- Exporting to the **window** namespace
  - If you want to make ANY function in your module available to the browser (and any other non module JS file), you must to export them to the **window** namespace

```
function initDatabase(){ …  }
…
window.initDatabase= initDatabase;
```

  - **otherwise they will not be available elsewhere**

For example

- If we did not declare initDatabase as exported and in index.js (which is not a module) we called

```
function submitForm(){
    document.getElementById('results').innerHTML='';
    initDatabase();
…
```

- we would have an error in the JS console:
  - ***initDatabase undefined***
- The same would happen if we called in the HTML file, e.g.
  - ***<button onclick=" initDatabase()">***

If you do not declare your file as a module

- if we declared database.js as a normal JS file

```
<script src="/javascripts/database.js"></script>
```

  - rather than a module

```
<script src="/javascripts/database.js" type="module"></script>
```

  - and in that file we included a module

```
import * as idb from './idb/index.js';
```

  - we would get an error telling us that you cannot import outside a module

  - please keep in mind these two errors as you will end uno having them often if you have not clear the concept of namespaces

Back to our programme

- Our EJS file includes two main JS files
  - one is a module, as it accesses the idb module (so it must be a module itself)

```
<script src="/javascripts/database.js" type="module"></script>
<script src="/javascripts/index.js" ></script>
```


In index.js

```
function submitForm(){
    document.getElementById('results').innerHTML='';
    let data= serialiseForm();
    let sum= parseInt(data.no1)+parseInt(data.no2);
    // store the sum to the DB
    storeSumData({no1: data.no1, no2: data.no2, sum: sum});
    // and now it retrieves all the sums that have given the same result
    getSumData(sum);
}
```

StoreSumData

- Is asynchronous because it accesses the IndexDB
- it inits the database in case it has not been initialised
- it creates a transaction and accesses the store via the transaction
- it inserts the data structure
- waits for the transaction to complete
- note: it is exported into the window namespace
  - as it is part of database.js which a module


```
async function storeSumData(sumObject) {
    console.log('inserting: '+JSON.stringify(sumObject));
    if (!db)
        await initDatabase();
    if (db) {
        try{
            let tx = await db.transaction(SUMS_STORE_NAME, 'readwrite');
            let store = await tx.objectStore(SUMS_STORE_NAME);
            await store.put(sumObject);
            await  tx.complete;
            console.log('added item to the store! '+ JSON.stringify(sumObject));
        } catch(error) {
            console.log('error: I could not store the element. Reason: '+error);
        };
    }
    else localStorage.setItem(sumObject.sum, JSON.stringify(sumObject));
}
window.storeSumData= storeSumData;
```


- inspect the entire code and make sure you understand it very well
  - then move to the exercise

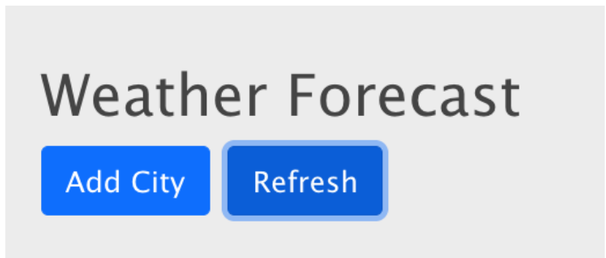
Off you go

Any questions we are here to answer

Week 4.b Indexed DB the Exercise

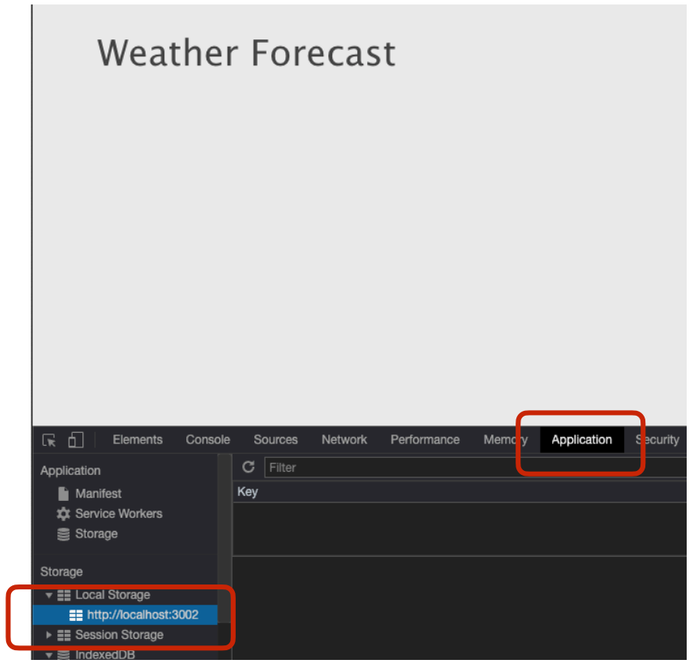
Prof. Fabio Ciravegna
Department of Computer Science
The University of Sheffield
f.ciravegna@shef.ac.uk

- You are given a client server architecture that retrieves data from the server about weather conditions in different cities
  - it runs on port 3002
- initially the interface will be empty
  - note if it is not empty,
    - clear the browser's local storage
      — see next slide



if you need to delete the data in local storage

- So to make sure there is no interference when moving to IndexedDB
- Right click on local storage > http.. and select clear



Click plus to add cities

- The data is cached using Javascript's local storage
  - e.g. in the file database.js

```
function storeCachedData(city, forecastObject) {
    localStorage.setItem(city, JSON.stringify(forecastObject));
}
window.storeCachedData= storeCachedData;
```

  - Note: the file database.js is defined as a module as it provides a virtual database implemented using local storage
- The exercise is to replace the database based on local storage and implement it via IndexedDB

Steps

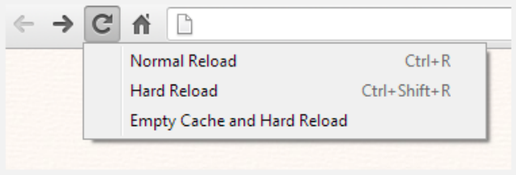
- the idb library is already available in the scripts folder
- add the initialisation of the database in the function initWeatherForecasts() in app.js
  - which is executed at loading time
- replace all the functions using local storage with equivalent functions using Indexed DB
  - note: all the functions should be async

For example

```
async function storeCachedData(city, forecastObject) {
    console.log('inserting: '+JSON.stringify(forecastObject));
    if (!db)
        await initDatabase();
    if (db) {
        try{
            let tx = await db.transaction(FORECAST_STORE_NAME, 'readwrite');
            let store = await tx.objectStore(FORECAST_STORE_NAME);
            await store.put(forecastObject);
            await  tx.complete;
            console.log('added item to the store! '+ JSON.stringify(forecastObject));
        } catch(error) {
            localStorage.setItem(city, JSON.stringify(forecastObject));
        };
    }
    else localStorage.setItem(city, JSON.stringify(forecastObject));
}
window.storeCachedData= storeCachedData;
```

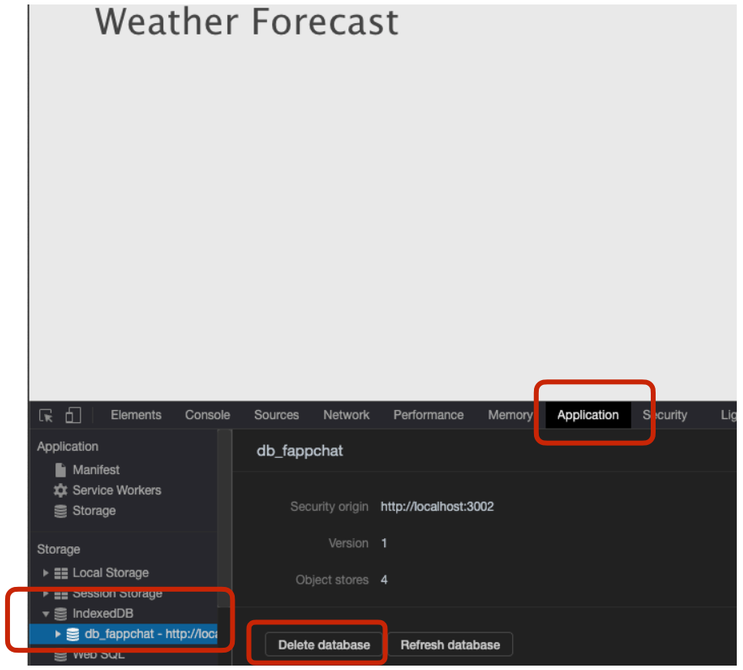

Some important issues

- After changing the code, force reload your files
  - long press chrome's reload button and select empty cache and hard reload



To delete the data from IndexedDB

- During testing you will need to delete your database to start afresh
- Open Chrome's debugger



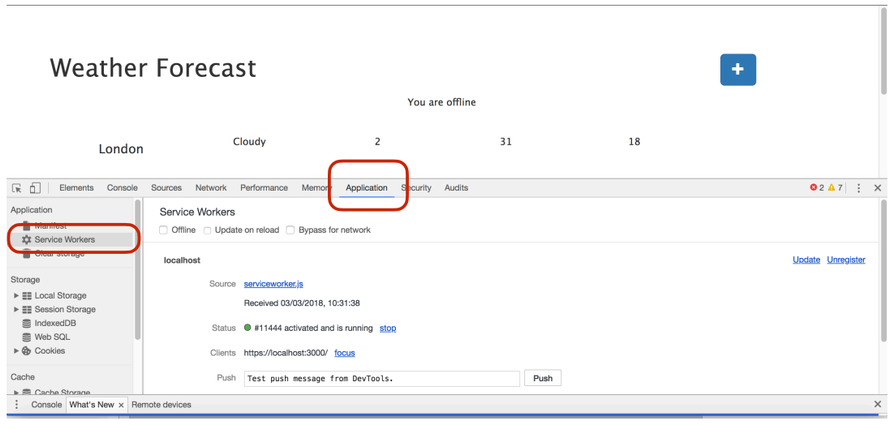
Any questions, just ask

Debugging Your Service Worker

Prof. Fabio Ciravegna
Department of Computer Science
The University of Sheffield
f.ciravegna@shef.ac.uk

Debugging

- Open Chrome development tools for your web app



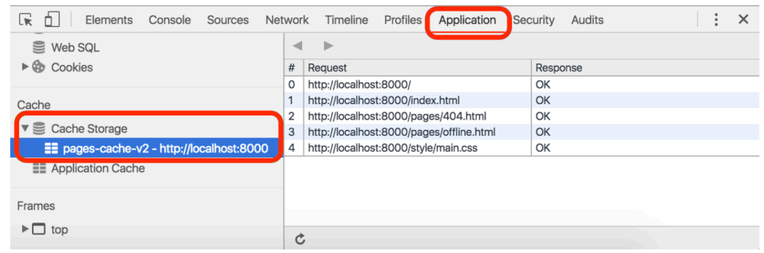
Inspecting cache

Inspect cache storage

Check the service worker cache

Chrome

Open **DevTools** and select the **Application** panel. In the navigation bar click **Cache Storage** to see a list of caches. Click a specific cache to see the resources it contains.

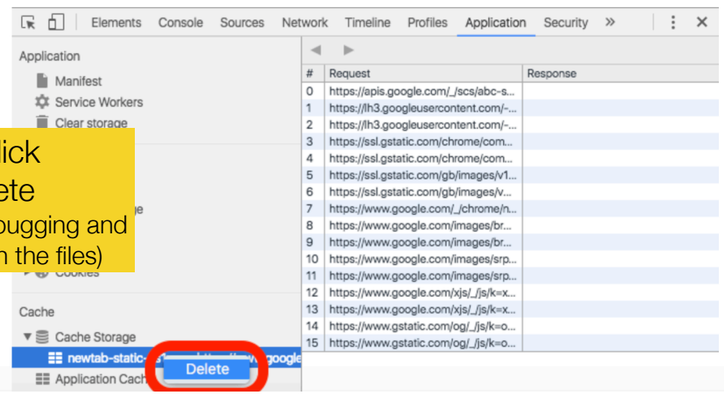


Clear Cache

Clear the service worker cache

Chrome

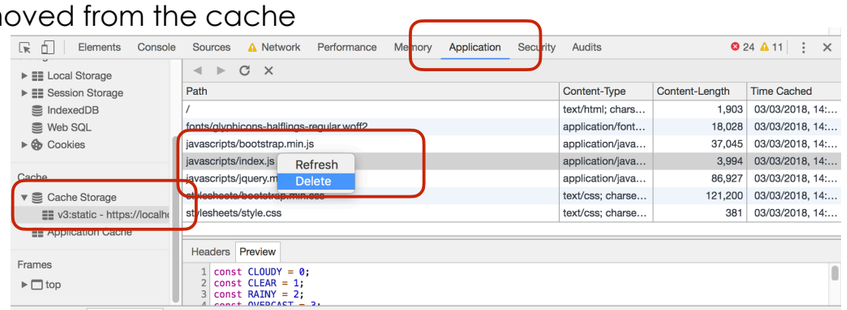
Go to **Cache Storage** in **DevTools**. In the **Application** panel, expand **Cache Storage**. Right-click the cache name and then select **Delete**.



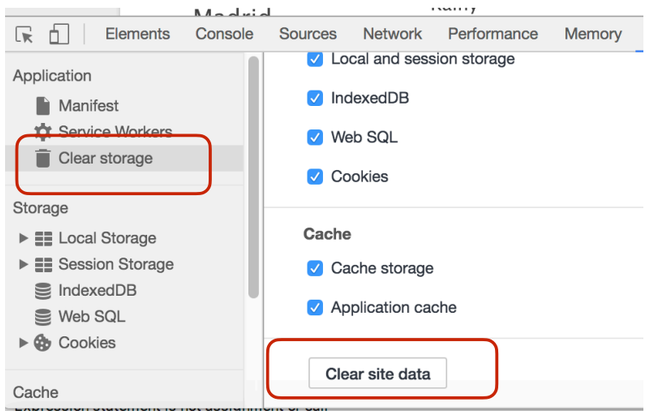
right click
to delete
(useful when debugging and changing often the files)

Modifying cached files

- To modify a cached file during development
  - modify the file
  - open chrome tools, choose the cache and right click on file to reload
  - then set the worker online again (application - Service worker - untick offline - otherwise you will get an error on loading the file you have just removed from the cache

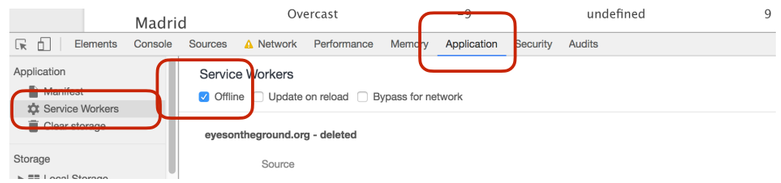


clear all your data



always check for errors

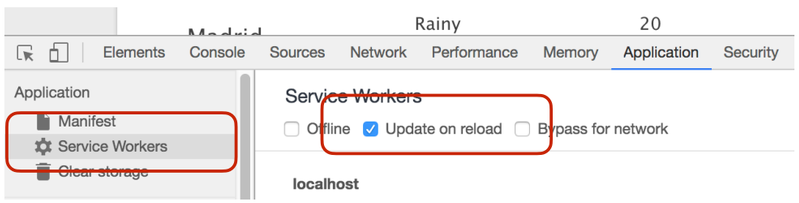
- if your service worker reports an error at installation time, it will not ACTIVATE
  - if it does not install **and** activate, it is not working, so you are effectively getting the data from the network
    - to test:
      - make sure to check the output of the console showing the activation
      - try putting some break points - it must stop when you reload the page
      - test the site via the dev tools



Remember to reload worker

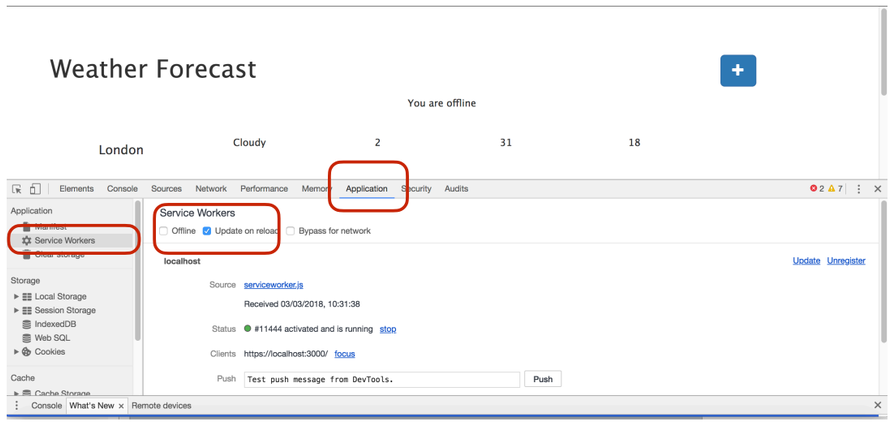
If you modify the code of the service worker, always remember to

1. tick update on reload (after loading remember to untick! !important)
2. **close all the tabs** pointing to any of the pages controlled by the worker (!important) or the old web worker will be alive
3. reload the page

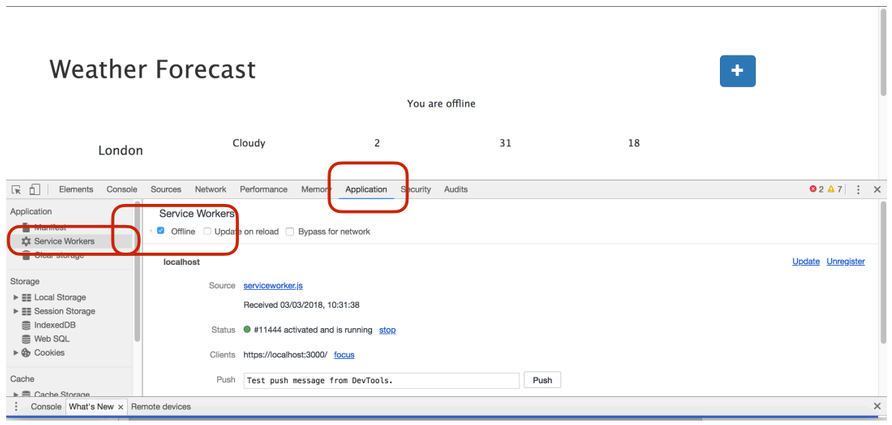


Updating service workers

https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#update



Testing offline



NOTE

- When you are offline, Chrome will print out errors for your attempts to go to the network
  - if you have an Ajax call, you must have a fallback position

Lab Class Week 4: Service Workers

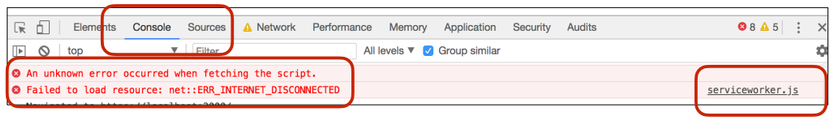
Prof. Fabio Ciravegna
University of Sheffield
f.ciravegna@shef.ac.uk

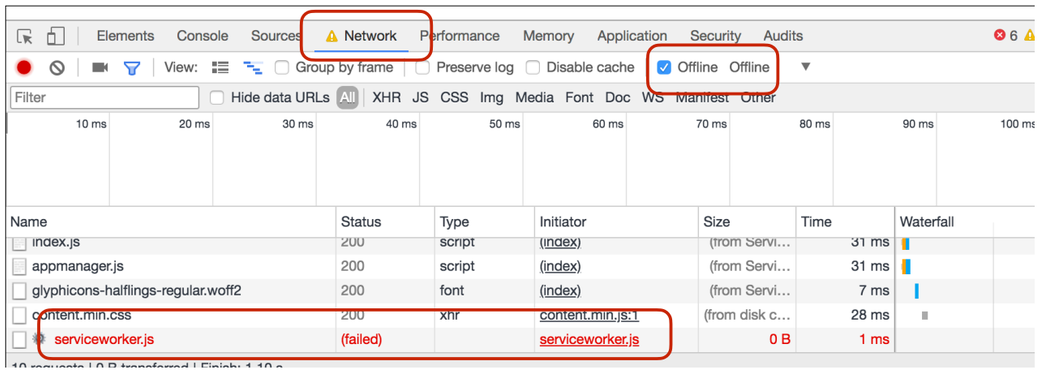
Exercises

- Download the app provided:
- It is the same we used in the Indexed DB exercise which retrieves weather forecasts from the server
  - It is the starting point
    - the one using local storage
    - with an added service worker

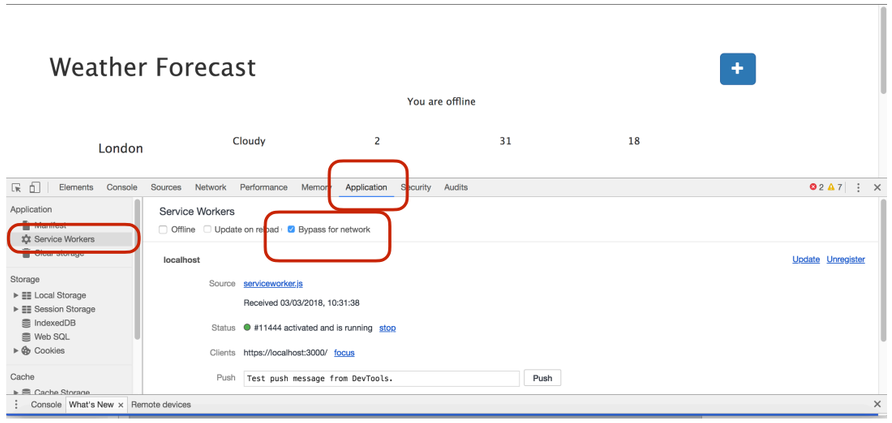
Do not worry if…

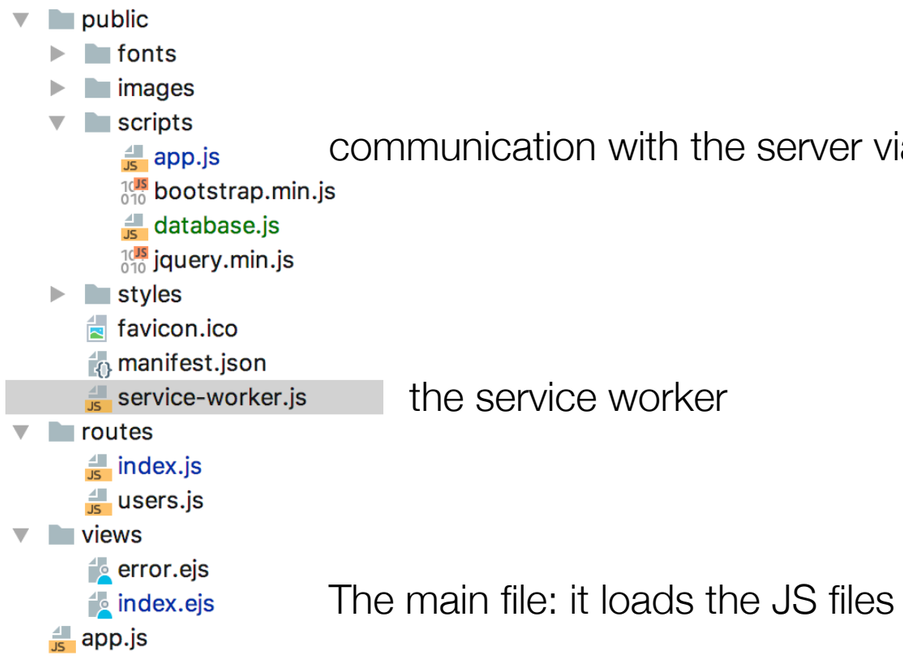
- When offline the page first tries to load a new version of the service worker from network and cannot fetch it. It is the correct behaviour





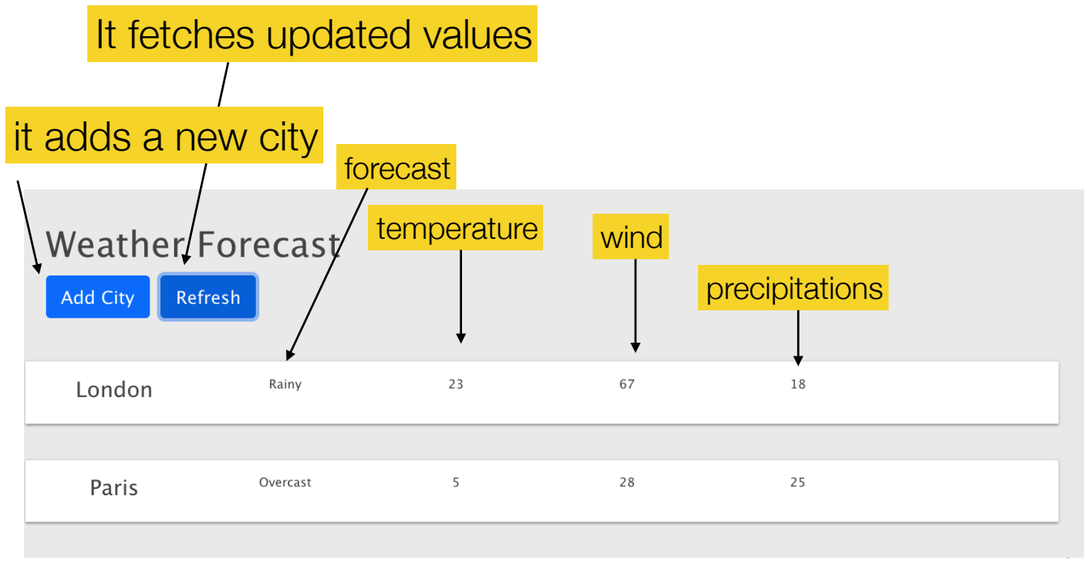
Testing network only





The exercises

- Goal of the exercise is to understand how a service worker works
- You will be asked to control the interplay between
  - requests made to the server
  - requests dealt with by the service worker



Exercise 1

- Inspect the app carefully and make sure to understand its parts
- Debug
  - the service worker (using Chrome developers tools)
  - the client's Axios call (using Chrome developers tools)
  - the server (using WebStorm)
- You must make sure to understand how it works
- Check the app both online and offline
  - see slides on how to debug the service worker

Exercise 2

- Add the service worker to the Indexed DB version of the app
  - using the solution you provided or the one I provided
- This will require you to understand the code and reproduce it
- Important note!!!
  - always remember to copy the files locally
    - e.g. bootstrap's files
  - or the system will not work offline

Exercise 3 (optional)

- Modify the service worker to implement the Stale-while-revalidate strategy
  - which returns the cached page but also updates the cash using the network
- See slide 35-36 of the lecture

Any issues just ask

The solutions will be available at 10am