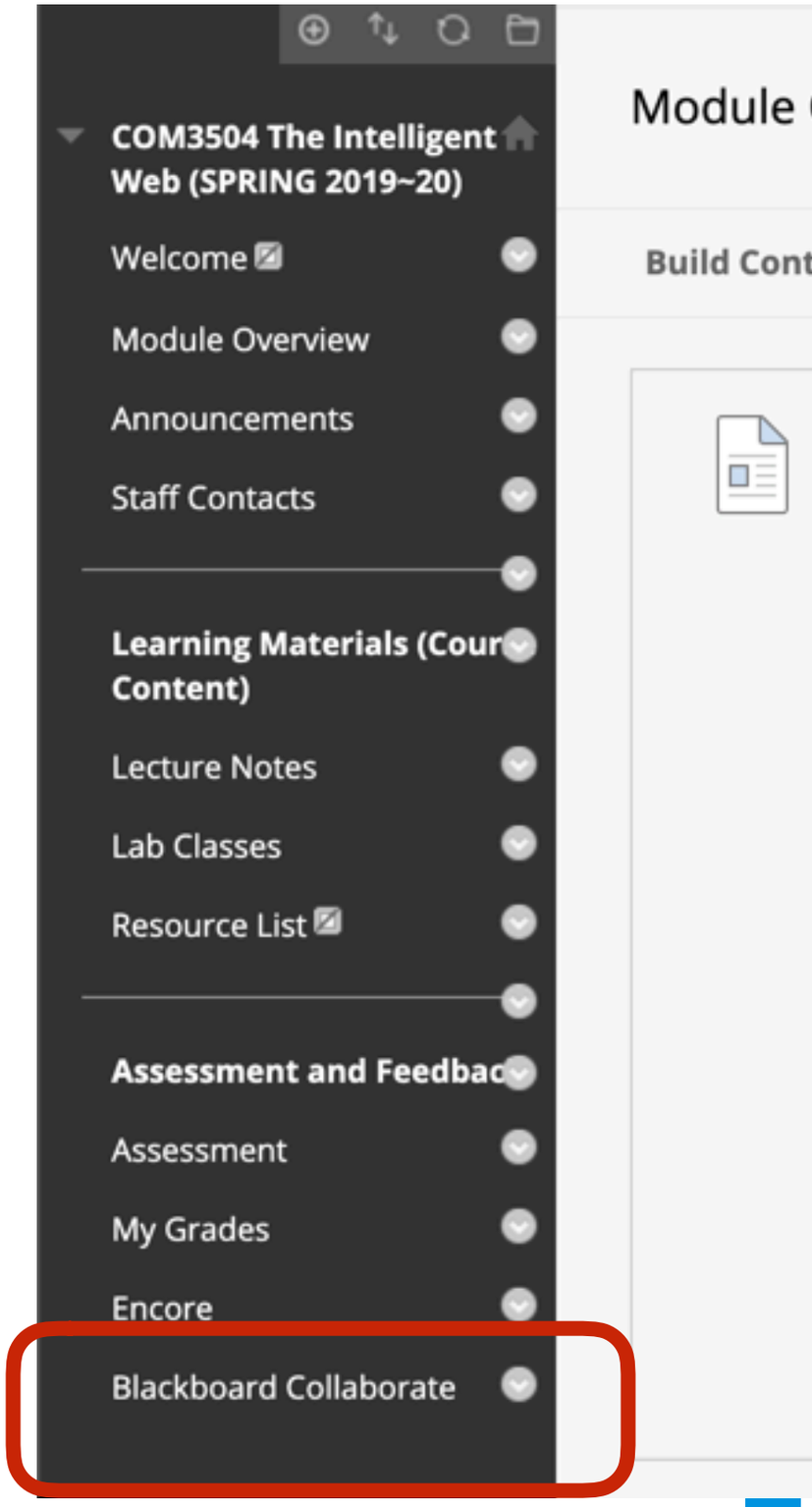
Week 5: IndexedDB lab class

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

How we will work today

- Running the lab remotely will be challenging
- I want to ensure that you can ask questions while you do the exercise
- We will use BlackBoard Collaborate



The University Of Sheffield.

COM3504 The Intelligent Web (SPRING 2019~20)

- Welcome
- Module Overview
- Announcements
- Staff Contacts

Learning Materials (Cour Content)

- Lecture Notes
- Lab Classes
- Resource List

Blackboard Collaborate Ultra

Sessions

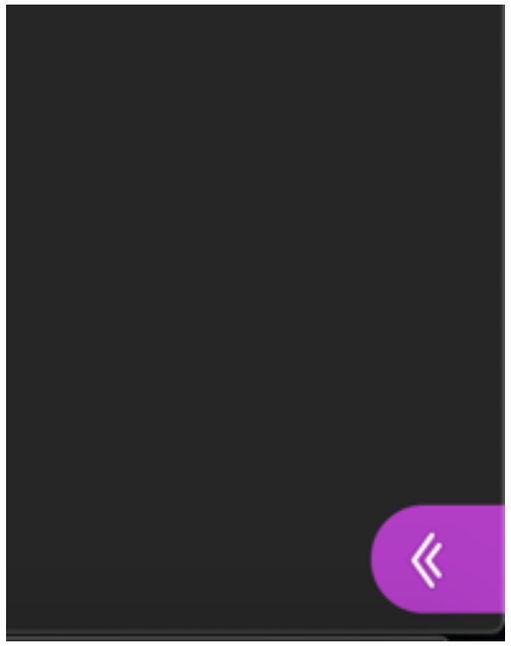
COM3504 The Intelligent Web (SPRING 2019~20) – Course Room
Unlocked (available)

Create Session

COM3504/6504 Lecture and Lab session
20/03/2020, 11:00 – 20/03/2020, 14:00 (not yet started)

How to ask questions

- Please do the exercise. If you have any question you can use collaborate in this way

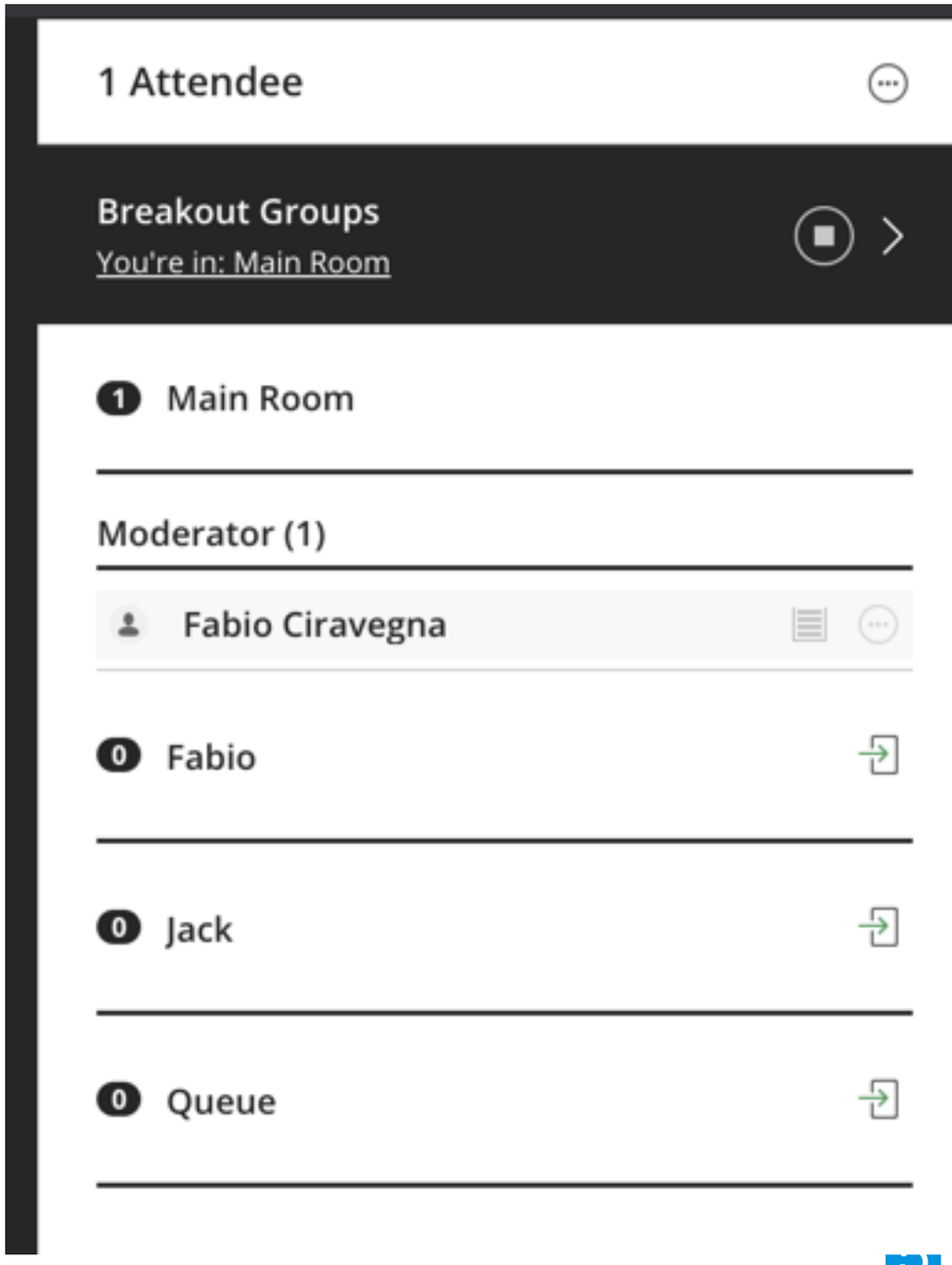


- Click on bottom right corner

The University Of Sheffield.

- Click on breakout groups

- you should see:



- Add yourself to the group
- When one of us will be available, we will move you to the appropriate room where we should be able to screen share and chat

Note

- We have never used it before so glitches may happen
- Apologies in advance if this happen
  - these are special times

The exercise

Aims of the exercise

- The aim of the class is to learn to use IndexedDB to store data on the client
- Download the solution to last week's exercise and replace the use of localStorage with IndexedDB to store the data locally
- In particular, you will have to change the following methods:
  - `initWeatherForecasts` in app.js to initialise the IndexedDB
  - `storeCachedData` in database.js to store the data in IndexedDB rather than in localStorage
  - `getCachedData` in database.js to retrieve the data from IndexedDB rather than from localStorage

Add the library

- Remember to add the library `Indexeddb-promised` to your module
- Download the file ids.js from Mole

- This comes from

- https://github.com/jakearchibald/indexeddb-promised
  - however this has recently changed and the new module is more difficult to integrate so I prefer to use the old version
    - if you use the new version from github, note that some of the methods names have changed (e.g. open-<openDB)

.

to add the .js file

- add it to the script folder under public
- add a link to it in the index.ejs file so that it is loaded
- add it to the cache of the service worker

How do you do that?

- In the initialisation:
  - check that your browser supports IndexedDB
  - initialise the database (creation of stores - including providing the appropriate indexes - what/how do you want to retrieve?)
  - Relevant code from slides

```
(function() {
  'use strict';   // impose strict syntax checks on the Javascript code

  //check for support
  if (!('indexedDB' in window)) {
    console.log('This browser doesn\'t support IndexedDB');
    return;
  }

  var dbPromise = idb.open('test-db2', 1,
     function(upgradeDb) {
       console.log('making a new object store');
       if (!upgradeDb.objectStoreNames.contains('firstOS')) {
         upgradeDb.createObjectStore('firstOS');
       }
     });
})();
```

The data schema

- IndexedDB is schema-less
  - it stores Javascript Objects
  - what will we store?
  - hint: in last week's solution we stored the forecast for each city as a Javascript object indexed by city name
    - Is it still a good solution?
    - To answer that question you will have to understand what you use the data for
      - i.e. how will you retrieve it?
      - has anything changed from last week?

Storing data

- The method receives the forecasts for one city at a time

```
function storeCachedData(city, forecastObject)
```

- you must store the object in your city store
- relevant code from the lecture slides

```
dbPromise.then(async db => {  // async is necessary as we use await below
    var tx = db.transaction('store', 'readwrite');
    var store = tx.objectStore('store');
    var item = {
      name: 'sandwich',
      price: 4.99,
      description: 'A very tasty sandwich',
      created: new Date().getTime()
    };
    await store.add(item); //await necessary as add return a promise
    return tx.complete;
}).then(function () {
    console.log('added item to the store! '+ JSON.stringify(item));
}).catch(function (error) {
    //do something
});
Question: shall we really use add? Is there a better alternative?
```

Retrieving the data

- How do we retrieve the data?
  - city by city
- Relevant code from slides

```
function getLoginData(loginObject) {
    if (dbPromise) {
        dbPromise.then(function (db) {
            console.log('fetching: '+login);
            var tx = db.transaction(LOGIN_STORE_NAME, 'readonly');
            var store = tx.objectStore(LOGIN_STORE_NAME);
            var index = store.index('userId');
            return index.get(IDBKeyRange.only(loginObject.userId));
        }).then(function (foundObject) {
            if (foundObject && (foundObject.userId==loginObject.userId &&
foundObject.password==loginObject.password)){
                console.log(“login successful”);
            } else {
                alert("login or password incorrect")
            }
        });
    }
}
```

However (advanced users)

- There is a complication
  - The city data is stored with the date it was retrieved on
  - So there is not just one element for the city
  - You will have to retrieve the most recent forecast
    - Solutions:
    - 1.
      - use `put` instead of `add` and do not store the date (bad idea)
    - 2.
      - store the city data inclusive of the date
      - getAll all the values for a specific city
      - cycle on the values retrieved and select the highest date

The University Of Sheffield.

- Relevant code for solution 2

```
function getAllDataAboutAValue(aValue) {
    if (dbPromise) {
        dbPromise.then(function (db) {
            console.log('fetching: ‘+aValue+ ‘ from
database’);
            var tx = db.transaction(STORE_NAME, 'readonly');
            var store = tx.objectStore(STORE_NAME);
            var index = store.index('someindex');
            return index.getAll(IDBKeyRange.only(aValue));
        }).then(function (itemsList) {
            if (itemsList && itemsList.length>0){
                // cycle here on all values and select the
                // element with the highest value

    …
    }) …
```

Off you go

Any questions we are here to answer