gh-pages to-do-notifications / scripts / todo.js / Jump to Go to file ...

chrisdavidmills adding in feature detection to allow Safari to use permissions reques... ... Latest commit 8b3e170 on Dec 11, 2019 History

1 contributor

416 lines (334 sloc) 16.6 KB Raw Blame

```
1  // create a reference to the notifications list in the bottom of the app; we will write database messages into this list by
2  //appending list items on to the inner HTML of this variable - this is all the lines that say note.innerHTML += '<li>foo</li>';
3  const note = document.getElementById('notifications');
4
5  // create an instance of a db object for us to store the IDB data in
6  let db;
7
8  // create a blank instance of the object that is used to transfer data into the IDB. This is mainly for reference
9  let newItem = [
10       { taskTitle: "", hours: 0, minutes: 0, day: 0, month: "", year: 0, notified: "no" }
11     ];
12
13 // all the variables we need for the app
14 const taskList = document.getElementById('task-list');
```

```
15
16  const taskForm = document.getElementById('task-form');
17  const title = document.getElementById('title');
18
19  const hours = document.getElementById('deadline-hours');
20  const minutes = document.getElementById('deadline-minutes');
21  const day = document.getElementById('deadline-day');
22  const month = document.getElementById('deadline-month');
23  const year = document.getElementById('deadline-year');
24
25  const submit = document.getElementById('submit');
26
27  const notificationBtn = document.getElementById('enable');
28
29  // Do an initial check to see what the notification permission state is
30
31  if(Notification.permission === 'denied' || Notification.permission === 'default') {
32    notificationBtn.style.display = 'block';
33  } else {
34    notificationBtn.style.display = 'none';
35  }
36
37  window.onload = function() {
38    note.innerHTML += '<li>App initialised.</li>';
39    // In the following line, you should include the prefixes of implementations you want to test.
40    window.indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB;
41    // DON'T use "var indexedDB = ..." if you're not in a function.
42    // Moreover, you may need references to some window.IDB* objects:
43    window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
44    window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange || window.msIDBKeyRange;
45    // (Mozilla has never prefixed these objects, so we don't need window.mozIDB*)
46
47    // Let us open our database
48    const DBOpenRequest = window.indexedDB.open("toDoList", 4);
49
50    // Gecko-only IndexedDB temp storage option:
51    // var request = window.indexedDB.open("toDoList", {version: 4, storage: "temporary"});
52
53    // these two event handlers act on the database being opened successfully, or not
54    DBOpenRequest.onerror = function(event) {

55      note.innerHTML += '<li>Error loading database.</li>';
56    };
```

```
57
58  DBOpenRequest.onsuccess = function(event) {
59    note.innerHTML += '<li>Database initialised.</li>';
60
61    // store the result of opening the database in the db variable. This is used a lot below
62    db = DBOpenRequest.result;
63
64    // Run the displayData() function to populate the task list with all the to-do list data already in the IDB
65    displayData();
66  };
67
68  // This event handles the event whereby a new version of the database needs to be created
69  // Either one has not been created before, or a new version number has been submitted via the
70  // window.indexedDB.open line above
71  //it is only implemented in recent browsers
72  DBOpenRequest.onupgradeneeded = function(event) {
73    let db = event.target.result;
74
75    db.onerror = function(event) {
76      note.innerHTML += '<li>Error loading database.</li>';
77    };
78
79    // Create an objectStore for this database
80
81    let objectStore = db.createObjectStore("toDoList", { keyPath: "taskTitle" });
82
83    // define what data items the objectStore will contain
84
85    objectStore.createIndex("hours", "hours", { unique: false });
86    objectStore.createIndex("minutes", "minutes", { unique: false });
87    objectStore.createIndex("day", "day", { unique: false });
88    objectStore.createIndex("month", "month", { unique: false });
89    objectStore.createIndex("year", "year", { unique: false });
90
91    objectStore.createIndex("notified", "notified", { unique: false });
92
93    note.innerHTML += '<li>Object store created.</li>';
94  };
95
96  function displayData() {

97    // first clear the content of the task list so that you don't get a huge long list of duplicate stuff each time
98    //the display is updated.
```

```
    taskList.innerHTML = "";

    // Open our object store and then get a cursor list of all the different data items in the IDB to iterate through
    let objectStore = db.transaction('toDoList').objectStore('toDoList');
    objectStore.openCursor().onsuccess = function(event) {
      let cursor = event.target.result;
        // if there is still another cursor to go, keep runing this code
        if(cursor) {
          // create a list item to put each data item inside when displaying it
          const listItem = document.createElement('li');

          // check which suffix the deadline day of the month needs
          if(cursor.value.day == 1 || cursor.value.day == 21 || cursor.value.day == 31) {
            daySuffix = "st";
          } else if(cursor.value.day == 2 || cursor.value.day == 22) {
            daySuffix = "nd";
          } else if(cursor.value.day == 3 || cursor.value.day == 23) {
            daySuffix = "rd";
          } else {
            daySuffix = "th";
          }

          // build the to-do list entry and put it into the list item via innerHTML.
          listItem.innerHTML = cursor.value.taskTitle + ' – ' + cursor.value.hours + ':' + cursor.value.minutes + ', ' + cursor.value.month +

          if(cursor.value.notified == "yes") {
            listItem.style.textDecoration = "line-through";
            listItem.style.color = "rgba(255,0,0,0.5)";
          }

          // put the item item inside the task list
          taskList.appendChild(listItem);

          // create a delete button inside each list item, giving it an event handler so that it runs the deleteButton()
          // function when clicked
          const deleteButton = document.createElement('button');
          listItem.appendChild(deleteButton);
          deleteButton.innerHTML = 'X';
          // here we are setting a data attribute on our delete button to say what task we want deleted if it is clicked!
          deleteButton.setAttribute('data-task', cursor.value.taskTitle);

          deleteButton.onclick = function(event) {
            deleteItem(event);
```

```
        }

        // continue on to the next item in the cursor
        cursor.continue();

      // if there are no more cursor items to iterate through, say so, and exit the function
      } else {
        note.innerHTML += '<li>Entries all displayed.</li>';
      }
    }
  }

// give the form submit button an event listener so that when the form is submitted the addData() function is run
taskForm.addEventListener('submit',addData,false);

function addData(e) {
  // prevent default - we don't want the form to submit in the conventional way
  e.preventDefault();

  // Stop the form submitting if any values are left empty. This is just for browsers that don't support the HTML5 form
  // required attributes
  if(title.value == '' || hours.value == null || minutes.value == null || day.value == '' || month.value == '' || year.value == null) {
    note.innerHTML += '<li>Data not submitted — form incomplete.</li>';
    return;
  } else {

    // grab the values entered into the form fields and store them in an object ready for being inserted into the IDB
    let newItem = [
      { taskTitle: title.value, hours: hours.value, minutes: minutes.value, day: day.value, month: month.value, year: year.value, notified:
    ];

    // open a read/write db transaction, ready for adding the data
    let transaction = db.transaction(["toDoList"], "readwrite");

    // report on the success of the transaction completing, when everything is done
    transaction.oncomplete = function() {
      note.innerHTML += '<li>Transaction completed: database modification finished.</li>';

      // update the display of data to show the newly added item, by running displayData() again.
      displayData();

    };
```

```
      transaction.onerror = function() {
        note.innerHTML += '<li>Transaction not opened due to error: ' + transaction.error + '</li>';
      };

      // call an object store that's already been added to the database
      let objectStore = transaction.objectStore("toDoList");
      console.log(objectStore.indexNames);
      console.log(objectStore.keyPath);
      console.log(objectStore.name);
      console.log(objectStore.transaction);
      console.log(objectStore.autoIncrement);

      // Make a request to add our newItem object to the object store
      let objectStoreRequest = objectStore.add(newItem[0]);
        objectStoreRequest.onsuccess = function(event) {

          // report the success of our request
          // (to detect whether it has been succesfully
          // added to the database, you'd look at transaction.oncomplete)
          note.innerHTML += '<li>Request successful.</li>';

          // clear the form, ready for adding the next entry
          title.value = '';
          hours.value = null;
          minutes.value = null;
          day.value = 01;
          month.value = 'January';
          year.value = 2020;

        };

      };

    };

  function deleteItem(event) {
    // retrieve the name of the task we want to delete
    let dataTask = event.target.getAttribute('data-task');

    // open a database transaction and delete the task, finding it by the name we retrieved above

    let transaction = db.transaction(["toDoList"], "readwrite");
    let request = transaction.objectStore("toDoList").delete(dataTask);
```

```
225
226     // report that the data item has been deleted
227     transaction.oncomplete = function() {
228       // delete the parent of the button, which is the list item, so it no longer is displayed
229       event.target.parentNode.parentNode.removeChild(event.target.parentNode);
230       note.innerHTML += '<li>Task \"' + dataTask + '\" deleted.</li>';
231     };
232   };
233
234   // this function checks whether the deadline for each task is up or not, and responds appropriately
235   function checkDeadlines() {
236     // First of all check whether notifications are enabled or denied
237     if(Notification.permission === 'denied' || Notification.permission === 'default') {
238       notificationBtn.style.display = 'block';
239     } else {
240       notificationBtn.style.display = 'none';
241     }
242
243     // grab the time and date right now
244     const now = new Date();
245
246     // from the now variable, store the current minutes, hours, day of the month (getDate is needed for this, as getDay
247     // returns the day of the week, 1-7), month, year (getFullYear needed; getYear is deprecated, and returns a weird value
248     // that is not much use to anyone!) and seconds
249     const minuteCheck = now.getMinutes();
250     const hourCheck = now.getHours();
251     const dayCheck = now.getDate();
252     const monthCheck = now.getMonth();
253     const yearCheck = now.getFullYear();
254
255     // again, open a transaction then a cursor to iterate through all the data items in the IDB
256     let objectStore = db.transaction(['toDoList'], "readwrite").objectStore('toDoList');
257     objectStore.openCursor().onsuccess = function(event) {
258       let cursor = event.target.result;
259         if(cursor) {
260
261         // convert the month names we have installed in the IDB into a month number that JavaScript will understand.
262         // The JavaScript date object creates month values as a number between 0 and 11.
263         switch(cursor.value.month) {
264           case "January":
265             var monthNumber = 0;
266             break;
```

```
          case "February":
            var monthNumber = 1;
            break;
          case "March":
            var monthNumber = 2;
            break;
          case "April":
            var monthNumber = 3;
            break;
          case "May":
            var monthNumber = 4;
            break;
          case "June":
            var monthNumber = 5;
            break;
          case "July":
            var monthNumber = 6;
            break;
          case "August":
            var monthNumber = 7;
            break;
          case "September":
            var monthNumber = 8;
            break;
          case "October":
            var monthNumber = 9;
            break;
          case "November":
            var monthNumber = 10;
            break;
          case "December":
            var monthNumber = 11;
            break;
          default:
          alert('Incorrect month entered in database.');
        }
          // check if the current hours, minutes, day, month and year values match the stored values for each task in the IDB.
          // The + operator in this case converts numbers with leading zeros into their non leading zero equivalents, so e.g.
          // 09 -> 9. This is needed because JS date number values never have leading zeros, but our data might.
          // The secondsCheck = 0 check is so that you don't get duplicate notifications for the same task. The notification

          // will only appear when the seconds is 0, meaning that you won't get more than one notification for each task
          if(+(cursor.value.hours) == hourCheck && +(cursor.value.minutes) == minuteCheck && +(cursor.value.day) == dayCheck && monthNumber =
```

```
309
310           // If the numbers all do match, run the createNotification() function to create a system notification
311           // but only if the permission is set
312
313           if(Notification.permission === 'granted') {
314             createNotification(cursor.value.taskTitle);
315           }
316         }
317
318         // move on and perform the same deadline check on the next cursor item
319         cursor.continue();
320       }
321
322   }
323
324 }
325
326
327 // askNotificationPermission function to ask for permission when the "Enable notifications" button is clicked
328
329 function askNotificationPermission() {
330   // function to actually ask the permissions
331   function handlePermission(permission) {
332     // Whatever the user answers, we make sure Chrome stores the information
333     if(!('permission' in Notification)) {
334       Notification.permission = permission;
335     }
336
337     // set the button to shown or hidden, depending on what the user answers
338     if(Notification.permission === 'denied' || Notification.permission === 'default') {
339       notificationBtn.style.display = 'block';
340     } else {
341       notificationBtn.style.display = 'none';
342     }
343   }
344
345   // Let's check if the browser supports notifications
346   if (!"Notification" in window) {
347     console.log("This browser does not support notifications.");
348   } else {
349     if(checkNotificationPromise()) {
350       Notification.requestPermission()
```

```
        .then((permission) => {
          handlePermission(permission);
        })
      } else {
        Notification.requestPermission(function(permission) {
          handlePermission(permission);
        });
      }
    }
  }

  // Function to check whether browser supports the promise version of requestPermission()
  // Safari only supports the old callback-based version
  function checkNotificationPromise() {
    try {
      Notification.requestPermission().then();
    } catch(e) {
      return false;
    }

    return true;
  }

  // wire up notification permission functionality to "Enable notifications" button

  notificationBtn.addEventListener('click', askNotificationPermission);



  // function for creating the notification
  function createNotification(title) {

    // Create and show the notification
    let img = '/to-do-notifications/img/icon-128.png';
    let text = 'HEY! Your task "' + title + '" is now overdue.';
    let notification = new Notification('To do list', { body: text, icon: img });

    // we need to update the value of notified to "yes" in this particular data object, so the
    // notification won't be set off on it again


    // first open up a transaction as usual
    let objectStore = db.transaction(['toDoList'], "readwrite").objectStore('toDoList');
```

```
393
394     // get the to-do list object that has this title as it's title
395     let objectStoreTitleRequest = objectStore.get(title);
396
397     objectStoreTitleRequest.onsuccess = function() {
398       // grab the data object returned as the result
399       let data = objectStoreTitleRequest.result;
400
401       // update the notified value in the object to "yes"
402       data.notified = "yes";
403
404       // create another request that inserts the item back into the database
405       let updateTitleRequest = objectStore.put(data);
406
407       // when this new request succeeds, run the displayData() function again to update the display
408       updateTitleRequest.onsuccess = function() {
409         displayData();
410       }
411     }
412   }
413
414   // using a setInterval to run the checkDeadlines() function every second
415   setInterval(checkDeadlines, 1000);
416 }
```