Exercise 17 – Client-Side Storage

Objectives

To work with local and session storage to optimise an application.

Overview

In the first part of this exercise, you will be working with the SlideShow object you created in a previous chapter. We are going to download data from a simple web service, put the results into some client-side storage (exploring session and local storage) and use this on subsequent reloads of the application.

This exercise is scheduled to run for around **30 minutes**.

Exercise Set Up

1. In VSCode, open the file called **index.js** from the **Starter/src** folder inside **Ex17** of the **Exercises** folder.
2. This project has been set up to use Webpack and Babel, so from the *Ex17 Starter* folder, run the command **npm i** to install the required packages.
3. Fire up the application with the command **npm start**. It should open your browser and display the image gallery (with an error message).
4. Open another command window and navigate the prompt to **Ex17→Starter→Data**.
5. Run the command **json-server images.json**.
6. Refresh your browser to check if the gallery is working properly.

Part 1 - Session and Local Storage

Here we are going to have the initial request for data put into client-side storage. We are going to use session and then local storage, so we can examine the difference between them.

First of all, we need to populate the store with the data that is initially retrieved when we first load our application. This will happen within the **_ajaxLoadUsingAsync** function after we have our data.

Part 1.1 - Session Storage

1. After the declaration of **galleryData**, add an **if** statement to check to see if **sessionStorage** is available.

This is known as feature detection. The JavaScript we are about to write will only happen if the global window object has a property called sessionStorage.

2. If **sessionStorage** is available:
   a) Set a property of **images** on **sessionStorage** to be the **JSON.stringify** version of **galleryData**;
   b) Log out the new value of **sessionsStorage.images**.
3. Save you code and open the developer tools in the browser:
   a) Click on the **Application** tab.
   b) In the left-hand menu, click on **Session Storage** and then **localhost** to reveal what is in **sessionStorage** - this should be a *key/value pair*, where the *key* is **images** and the *value* is a *string representation of our array of image objects*.
   c) Check the *console* output also, it should be *a string containing our array of image objects*.
4. On the **Application** tab, right click on **localhost** and click **Clear**. This removes the data from **sessionStorage**.
5. Click on the **Network** tab and *refresh the page*. Observe that a call has been made to **images/**. Inspecting this confirms that the **_ajaxLoadUsingAsync** function has been called and an HTTP request made for the data.
6. Check back in **Application→Session Storage**. You should find that our data is back.

The next stage is to intercept the call to **_ajaxLoadUsingAsync** if **sessionStorage.images** already exists. That'll be one less HTTP request we need to make for data if it does.

7. Replace the code for the call to **_ajaxLoadUsingAsync** in the constructor with:
   a) An **if** statement that checks for **sessionStorage** *not* being present **OR** **sessionStorage.images** not being present and calls **_ajaxLoadUsingAsync** if this is the case;
   b) Otherwise, set **this_theGallery** to the **JSON.parse** version of **sessionStorage.images** and call **this._displayImages**.
8. *Repeat steps 4, 5 and 6 to ensure that* **sessionStorage** *is being populated*.
9. Open the **Network** tab and click *refresh* again and observe that the HTTP request to **images/** is no longer made as we are using **sessionStorage** to populate **_theGallery**!
10. With the **Network** tab open, navigate to another website and then back to **localhost:8080**.
    You should see that *no HTTP request* is made to **images/** as the session is still active and its data has persisted, even though we have left the site.
11. Quit the browser completely, reopen it (*not at* **localhost:8080**) and display the **Network** tab.
12. Navigate to **localhost:8080** looking at the requests made. You should observe that this time a call is made to **images/** as no Session Storage existed for this site.

Part 1.2 - Local Storage

Session Storage is only persistent whilst the browser remains open. Quitting the browser clears Session Storage, but what if we want to persist the data across a number of browser sessions. That's where Local Storage comes in.

1. In **index.js**, *replace all occurrences* of **sessionStorage** with **localStorage**.
2. In the browser, *clear all current Session Storage and Local Storage* by *right clicking* on the appropriate item and selecting **Clear**.

3. Open the **Network** tab and *refresh the browser*.

You should see a call to **images/** as there is no data in **localStorage** that matches what we want.

4. Switch back to the **Application** tab and observe that an **images** *key* has been added to Local Storage with a *value* of the *string representation of the array of image objects*.
5. Back on the **Network** tab, *refresh the page*.

There should be no call to **images/** as Local Storage already has the data we need.

6. Quit the browser, open the **Application** tab and observe that the **images** data is still available in Local Storage (even if we are not currently viewing our application).
7. Open the **Network** tab and *refresh the page*.

You should note that no request is made to images/ because the data is already present in Local Storage. Local Storage is persistent until cleared.