COMM049: Week 4 – More on JavaScript

Part I: Setting up the Shopping List Example

1. Download from Surrey Learn Week 4 shopping_list.zip and unpack it somewhere handy on your computer.
2. Log in to Cloud9 and open a new Workspace.
3. When the "Create a new workspace" dialogue opens, provide a name ("shopping_list", for example!). Leave "Public" selected, choose node.js for the template and finally click the green "Create workspace" button.
4. We will be lazy and use the server that Cloud9 sets up for us, but with a different index.html page. So the next thing is to delete the "index.html" file from the "/client" directory in the new Cloud9 workspace (right click and select "delete").
5. Now, click on "client" in the finder window and then select "File/Upload Local Files…" from the menu bar. This will open a dialogue to upload a file. Drag and drop the new "index.html" file from the folder you unpacked in Step 1 above.
6. Close the "Upload Files" dialogue.
7. Now click on the "js" folder in the finder, select "File/Upload Local Files…" from the menu bar again and this time drop in the "shopping.js" file that you downloaded from Step 1 (it is in the /js sub-folder). Close the dialogue.
8. Finally, click on the css folder and select "File/Upload Local Files…" from the menu bar. Drop in both normalize.css and shoppingInStyle.css from the /css subfolder. Close the dialogue. The first of those two files is not necessary if you use bootstrap but I suggest you just use some simple hand styling in this exercise without bootstrap.
9. You may notice that I have not included a "manifest" property in the opening html tag. This gets to be a bit of a nuisance when you are debugging an application so we leave it out for now and can put it back at the end.
10. You can run the project now. Actually you have two alternatives – you could either run the server.js file (go to the console window and type "node server.js", or you could click on the index.html and click the Run icon in the menu bar. In the second case Cloud9 will default to using Apache. If you use the first method then you can view the running application within Cloud9 by using the "Preview" menu item to select "Preview Running Application". Once you have checked it works there (if it doesn't, ask for help!), you could copy the address of the app and paste it into a browser window of your choice. If you use the second method the console window will show you the address for the running application. Again, copy it and paste it into a new browser window or tab.

11. Now play around a little bit and check to see (using your browser's developer tools, how Local and Session storage are updated as you interact with the app.

Part II: Some notes on building out the Shopping List example.

The first task is to associate a UserName with each Item. One way to do this is to store a JSON object in place of the simple Item. The code below has two modifications compared with the code that you set up in Part I. The first modification generates a new key for each item that is added to the list. The second modification creates a JSON object with two properties: name; and, item. That object must then be "serialised" before it is stored.

Serialisation is the process of converting a complex data structure into a format that enables it to be either stored, or transmitted across a network. This is done in a way that enables the data structure to be reconstructed when recovered from memory, or received by a peer in a network. Ideally, the data structure can be de-serialised/reconstructed in the same or in a different environment to which it was serialised.

With JSON, we "stringify" the object to serialise it. We "parse" the saved, or transmitted entity to de-serialise it.

In the first modification, we generate an integer that is equal to the length of the list of items in localStorage. This integer is, of course, incremented each time we add an item to localStorage. So we can generate a new key each time by simply concatenating "newItem" and the (string representation of the) integer.

N.B. You should always document the assumptions behind your design decisions. In this case, this assumes that the shopping list application is the only application that can add items into localStorage. This is fine, as localStorage is "sandboxed" to single domains (unless we serve multiple apps from the same domain).

It also assumes that we delete items from the shopping list in the reverse order to which they were entered. This is an acceptable assumption if the only way to delete items is through use of the clear function. But we probably want to allow users to remove entries from anywhere in the list. If we do this, then we will need to modify the implementation below into something a little more complex. Make sure you understand the reason for this. You should also have a think about how you could do this.

Anyway, for now the first modification to the shopping.js file is as shown below (changes in bold):

```
$(function() {
  var userName = prompt("Please enter your name");
  sessionStorage.setItem('userName', userName);

  $('#submitItem').click(function(event) {
    var i = localStorage.length;
    var key = 'newItem' + i;
    var item = $('#item').val();
    $('#item').val("");
    localStorage.setItem(key, item);
  })
})
```

As pointed out in last week's lecture, you could actually replace "key" in the third to last line with the expression that generates the value of "key" as we don't need to use this variable anywhere else. This would be more succinct, but possibly a little harder to read. It's a judgement call as to which way you prefer.

The next step is to create the JSON object that contains two properties (name and item) and the "stringify" it to enable it to be saved into localStorage. To do this, you need to modify the code as below (again, changed in bold):

```
$(function() {
  var userName = prompt("Please enter your name");
  sessionStorage.setItem('userName', userName);

$('#submitItem').click(function(event) {
    var i = localStorage.length;
    var key = 'newItem' + i;
    var item = $('#item').val();
    $('#item').val("");
    var namedItem = {};
      namedItem.name = sessionStorage.getItem('userName');
      namedItem.item = item;
    var jsonNamedItem = JSON.stringify(namedItem);
    localStorage.setItem(key, jsonNamedItem);
  })
})
```

Again, there are two more ways of generating the JSON object – check the lecture notes to make sure you understand these. The choice as to which method you use is really a matter of personal style. But it is best to pick one and stay with it (unless your organisation dictates a specific choice in its coding standards).

You now have two more modifications to try and make:

1) Modify the above click function on #submitItem so that the details of the item are appended to the shopping List table on the index page;
2) Once you have that working, add code into the "document ready" function, above, that fetches each entity that has been previously stored and adds it to the shopping list table so that the table is populated with data when the shoppingList app is first loaded. Remember that now when you getItem from localStorage you will need to parse it back into a JSON object (use JSON.parse( ) in an analogous way to the use of "stringify" above[1]).

I will upload the full solution so far later this week, but please do try to get it working on your own first.

Remember, remember: You will need to edit offline.appcache in order to force a reload of the cache. Despite Google's claims to "do no harm" the caching policies in Chrome can cause serious stress, especially on PCs it seems. If you are using Chrome, you may find it easier to debug the Shopping List application if you temporarily delete the "manifest" property from the opening <html> tag of your index.html page. This will stop offline working but that is not a problem while we build out the application: it's easy to put it back again later. (Actually, I already did that for you ☺).

If you complete the above, then there are some more tasks you may like to experiment with.

1. Add a button that enables all the Shopping Items to be cleared in one go. Remember, what I will want to see is just a button with an identifier in the html. Then in your JavaScript file, add a click event handler to this button that contains the functionality to clear all items from Local Storage.
2. Add a Delete Item button to each row of the table of items. Then in the JavaScript file, create a click event handler that deletes the corresponding item from local storage. A convention that is often used is that the text of the item is struck through (a line put through it) rather than removed from the list. See if you can do this as well (the idea is to use JavaScript to change the class of the corresponding row in the table and then in the CSS file define a class with the property { text-decoration: line-through; }). Then you will need to use the Local Storage API to remove the item. However, there is a problem. Remember that we used the number of items in the local store to determine the index for a new key. This is not going to work anymore so you will need a different method for creating

[1] var jsonNamedItem = JSON.stringify(namedItem);

the unique keys for new items. See if you can come up with a good strategy.

3. Finally, introduce media queries to ensure that the layout responds in an appropriate way to the device that is being used to view the app. Do something quite simple just to get a feel for this.